REC'D 03 DEC 2004
WIPO PCT

Kongeriget Danmark

Patent application No.: PA 2003 01635

Date of filing: 03 November 2003

Applicant: (Name and address)
MAT ApS
Fruebjergvej 3
DK-2100 København Ø
Denmark

Title: An electronic mathematical model builder

IPC: -

This is to certify that the attached documents are exact copies of the above mentioned patent application as originally filed.

Patent- og Varemærkestyrelsen Kongeriget Danmark

PRIORITY DOCUMENT
SUBMITTED OR TRANSMITTED IN
COMPLIANCE WITH
RULE 17.1(a) OR (b)

Patent- og Varemærkestyrelsen
Økonomi- og Erhvervsministeriet

19 November 2004

Susanne Morsing

BEST AVAILABLE COPY

PATENT- OG VAREMÆRKESTYRELSEN

AN ELECTRONIC MATHEMATICAL MODEL BUILDER

FIELD OF THE INVENTION

The present invention relates generally to the field of information processing by digital computers and, more particularly, to the entry and processing of information by application programs, particularly spreadsheets.

BACKGROUND OF THE INVENTION

The present description relies on technical terms used in connection with operation of a spreadsheet program, including spreadsheet formulas, for an overview see e.g. Microsoft Excel 2002 Bible, John Walkenbach, John Wiley & Sons Inc; ISBN: 0764535838.

Before computers, numerical analyses, particularly financial analyses, were usually prepared on an accountant's columnar pad or spreadsheet, with pencil and calculator in hand. By organizing data into columns and rows, spreadsheets afford the rapid assimilation of information by a reader. The task of preparing a spreadsheet on paper, however, is not quite so fast. Instead, the process tends to be very slow, as each entry must be tediously calculated and entered into the spreadsheet. Since all calculations are the responsibility of the person preparing the spreadsheet, manually prepared spreadsheets are also prone to errors. Hence, preparation of spreadsheets by hand is slow, tedious, and unreliable.

With the advent of microcomputers, a solution was forthcoming in the form of "electronic spreadsheets." Better known simply as "spreadsheets," these software programs provide a computerized replacement for the traditional financial modelling tools: The accountant's columnar pad, pencil, and calculator. The improvement represented by a spreadsheet program over these tools may be compared with the improvement provided by a word processor over a typewriter. Spreadsheets offer a dramatic improvement in ease of creating, editing, and using financial models.

A spreadsheet provides millions of cells grouped together in a single sheet or document. A single cell may contain a value, a label or a formula. In a conventional spreadsheet, millions of cells are provided even though only e.g. 25 cells are needed for a specific task. A conventional spreadsheet typically offers 256 columns and 65536 rows amounting to 16,777,216 cells. Most of the cells are not visible. A standard monitor typically displays about 500 cells at the time (33 rows by 15 columns). If a spreadsheet model covers more than 33 rows or 15 columns on a standard monitor, the user will either have to scroll up and down to manage or work

with the spreadsheet model or open different windows e.g. showing different parts of the model at the time.

Data is entered into an electronic spreadsheet in much the same way as an accountant enters information on his pad. After the user has positioned a screen cursor at a desired location (a cell), the user can enter alphanumeric information. Besides holding text and numerical information, however, spreadsheet cells can store special instructions or formulas specifying calculations to be performed on the data stored in spreadsheet cells. This is in fact one of the major disadvantages of conventional spreadsheets: Data and the user interface form a complex structure. For example, a single cell may contain a text-based formula hidden underneath the displayed numerical output value of the formula. Hence it is not easy for a spreadsheet user to document a spreadsheet model (i.e. describe to other users how to use a specific model and ensure that the model gives the correct result).

During creation of a particular spreadsheet model, a user enters formulas into cells the same way he or she enters values and labels.

Spreadsheet formulas are fundamental to the creation and operation of a spreadsheet.

A spreadsheet formula has a specific syntax defined by the spreadsheet designer. Typically, a formula begins with a special token (e.g. =, + or @) to distinguish the formula from raw data, followed by tokens as operators (e.g. + - /), values, cell references (e.g. =(B1-B2)/100), and/or built-in functions (e.g. = SUM (A1:A12)). Cell references or addresses can serve as variables in an equation, thereby allowing precise mathematical relationships to be defined between cells.

Before meaningful results of a given spreadsheet can be presented to a user, the formulas of a given spreadsheet model must be evaluated or recalculated. "Recalculation" is the process by which spreadsheet cells, particularly those holding formulas, are updated. In a conventional spreadsheet system, the standard recalculation order is termed "natural".

Suppose, for example, that it is desired to calculate the price of a good including VAT and shipping costs: The user enters the price of a good in cell A1, the VAT rate in cell B1 and the shipping rate in cell C1. The formula for VAT is entered in cell A4, shipping costs are calculated in cell B4, and total cost is calculated in cell A7, cf. Fig. 1.

By entering different values in cell A1, the user can calculate the total costs. Each

time the user enters a new value in cell A1 the spreadsheet system recalculates the model to show the result in cell A7. Interdependencies between the cells dictate how a spreadsheet system actually performs recalculation. Before the spreadsheet system can recalculate the formula in cell A7, the spreadsheet system must recalculate the formulas in the cells, A4 and B4. In the same way, the spreadsheet system must recalculate the formula in cell A4 before it can recalculate the formula in cell B4. This gives the order of recalculation: A4 -> B4 -> A7.

Users can use more than one built-in function in a single formula, with one built-in function constituting an argument or parameter to another built-in function. This is termed "nesting" built-in functions. Generally, a user can nest as many built-in functions as desired. For example, the nested function below calculates the average values in cell block C1:C14, and then rounds the average to two decimal places:

=ROUND (AVG (C1:C14);2)

Nested built-in functions evaluate from inside out. In the above example, for instance, the AVG is calculated before the ROUND.

When formula syntax and arguments are correct, most built-in functions and formulas recalculate to a numeric value (including dates) or a string value (label or text). If the user does not enter the formulas in a cell correctly, the result of a recalculation is a special value ERR (an error). It is therefore a precondition that the user of a spreadsheet system is familiar with the syntax of a spreadsheet formula. Different spreadsheet systems have different syntaxes to obey.

For example, if a spreadsheet user has to write the formula described in cell B4 in Fig. 1, the user has to know the syntax of an IF built-in function. (Three arguments divided by a special the token ';' or other argument separator (as specified by the implementation) and the meaning of the token '<=' not greater etc.).

With the present trend of employing electronic spreadsheets to model increasingly complex data models, conventional spreadsheets have disadvantages.

One disadvantage is related to the ability of most users to correctly enter and understand complex cell formulas using a conventional electronic spreadsheet system. Some formula-builder systems have been developed to address this problem, e.g. as disclosed in US 5,603,021. Formula-builders constitute a major improvement of spreadsheets and have certainly assisted the user in building a formula with the correct syntax. However, it has not solved the problem that most spreadsheet users find it difficult to manage the syntax of a spreadsheet formula.

Consequently the average spreadsheet user has difficulty in understanding the logic of a formula and hence the logic of the spreadsheet.

Another disadvantage is related to the ability to reuse formulas.

For example, the cell formula:

=ROUND (AVG (C1:C14); 2)

is constructed of two built-in functions and operates on the cellblock C1:C14. If it is desired to reuse this formula in a conventional spreadsheet, it must be copied and pasted to a different cell location either in the same spreadsheet or another spreadsheet that are currently residing in the memory of the computer. The copy and paste process will in turn adjust the relative cell references in the spreadsheet formula. If the formula is pasted into a cell located two columns to the right of the cell it was copied from, the cell references is adjusted to E1:E14. It is not possible to save the function as a built-in function under a name, e.g. ROUND_AVG_2 to for reuse on other cell references or as a nested function.

Typically, a financial spreadsheet includes a feature termed Goal Seek.

Consider a sales budget spreadsheet with Sales of 50000 in Cell B2, Costs of 45000 in Cell B3 and the formula =B2-B3 in Cell B4 to calculate gross profit. Further, cell B5 holds the formula =B4/B2 to calculate the gross profit percentage, cf. Fig. 2.

After a recalculation the result, 20% is displayed in cell B5. However, it is desired to reduce the costs to obtain a gross profit percentage of 25%. To obtain the wanted result, the spreadsheet user may enter random values manually into cell B3 until B5 attains the value 25%. Each time the user enters a new value in the cell the spreadsheet model recalculates. Another solution is to use the built-in Goal Seek tool, which is part of most modern spreadsheet systems. A typical Goal Seek tool has a dialog box where the user can enter specifications, cf. Fig. 3.

By entering the specified information in the dialog box and clicking the OK button, the Goal Seek tool starts an iterative process to find the specified value. The process resembles the process of finding the root in an equation. Various root-finding techniques are available, cf. e.g. William Press, Saul A Teukolsky, William T. Vetterling, and Brian P Flannery, Numerical Recipes in C. Cambridge University Press, second edition, 1992.

Regardless of the specific technique used in the spreadsheet system, the goal seek function will insert values in the specified "changing cell" until the desired condition is

fulfilled. The condition is fulfilled if the desired value is within a range of ± a delta value from the value specified in the "equal to cell". If the condition is not fulfilled, the goal seek function will stop after a maximum number of iterations (typical 500 iterations). Each time a new value is inserted in the "changing cell" by the goal seek tool the spreadsheet model is recalculated and the value in the "make cell" is compared with the value in the "equal to cell". Thus, the goal seek tool is external to the spreadsheet model meaning that the goal seek function is not recalculated each time the sales budget spreadsheet model is recalculated. If the goal seek tool was part of the spreadsheet model, the goal seek process (root finding) would restart each time the spreadsheet model was recalculated after a new value is inserted into the "changing cell", a process that would lead to an infinite loop. Instead the user must initiate the goal seek loop from a dialog box.

It is a disadvantage of conventional spreadsheets that the goal seek function must be external to the spreadsheet to avoid infinitive recalculation. Further, the inexperienced spreadsheet user finds it difficult to manage a spreadsheet model and specify the demanded input in the goal seek dialog box.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide a new calculation tool that overcomes the above-mentioned disadvantages.

According to the present invention, an electronic mathematical model builder that fulfils the above-mentioned and other objects, comprises

a memory for storage of data,

a processor for defining addressable sets of cells stored in the memory with a unique identifier, for entering data into the cells and for processing data stored in the cells,

a user interface with a display for displaying sets of cells in a work area and means for creating and positioning sets of cells in the work area and inputting data to be entered into the cells.

The electronic mathematical model builder may further comprise a function builder for building mathematical relations between cells, comprising fields for user specification of a desired function by mathematical operators and input variables of the function, and a set of destination cells containing values of the function.

It is an important advantage of the present invention that a function created with the function builder is not inherently connected to one or more specific cells as in.

conventional spreadsheets. On the contrary, the operator of the mathematical model builder is free to define input cells containing input values for the function and destination cells containing output values of the function. In a preferred embodiment of the invention, a new set of cells is created containing the output values of the function. Further, the function may be stored in the memory as an independent entity for later retrieval and possible editing, the editing including one or more of the following: Renaming, redefinition of cell addresses of input variables, redefinition of cell addresses of output variables, modification of the function, etc.

An electronic mathematical model builder according to the present invention may comprise features and functions of conventional spreadsheets, such as cell display format features, mathematical functions, charts, cell protection features, user interface features, such as icons, tools, tool bars, etc., etc.

For example, a cell may contain various types of data, such as a numerical value, an alphanumeric string, a logical value, a formula, etc.

Further, various features of a cell may be specified by a user, features such as character fonts used to display the contents of a cell, attributes of displayed characters, such as underline, overline, bold, italic, etc., justification in a cell of a string displayed, such as left, right, centre, etc., attributes of displayed borders, patterns, and colours of cells, etc.

Cells may be copied, pasted, moved, etc., either into the current model or into another model, and cells may be inserted into, deleted from, or cleared in the current model.

Graphical objects, such as lines, arrows, rectangles, ellipses, arcs, polygons, text boxes, etc., may be created in the model.

Charts, such as line charts, pie charts, bar charts, area charts, column charts, XY (scatter) charts, radar charts, 3D type charts, etc., based on cell data may be created.

Cells of a model may be displayed in two dimensions by means for displaying cells, such as a CRT monitor, a LCD monitor, a plasma monitor, a printer, etc., e.g. in a way well-known in the art of spreadsheets, i.e. as a grid-like table or matrix of cells holding data. In one embodiment, the grid-like table or matrix is formed of a series of columns intersecting a series of rows. The columns extend from a peripheral horizontal axis of the matrix while the rows extend from a peripheral vertical axis of the matrix. Each row is labelled with a numeral (1, 2, 3...) along the peripheral

vertical axis and each column is labelled with a letter (A, B, C,...) along the peripheral horizontal axis. An intersection of a row and column forms a cell, which receives a label formed by the composition of the respective labels of the specific row and column (e.g. C5).

In a preferred embodiment of the invention, cells are created and displayed in sets or panels. Each set or panel may be individually and independently positioned or moved around in the display of the model, e.g. using a mouse and a cursor to drag the set or panel in question to a desired position within the display of the model. A panel may hold one cell, a column of cells, a row of cells or a two-dimensional matrix of cells. The displayed panel may have row labels, column labels and a unique identifier. The panel identifier, its row number, and its column number uniquely identify each cell in the panel. The panel may further have a number of attributes defining features of the panel (e.g. the visibility of the row and column labels, the colour of the labels and cells, edibility of the cells, number format etc.).

In the illustrated embodiment of the present invention, the number of cells in a panel may be greater than the number of visible cells. The system provides means for the user to adjust the number of cells in a panel as well as the number of visible cells. Further, the system provides means for attaching a scrollbar to one or a group of panels with the purpose of scrolling cells outside the visible area into the displayed panel area.

In the function builder, cells providing input values to the function may be selected in a way similar to or identical to the way cells are selected in conventional spreadsheets, i.e. selecting desired cells using a pointing device, such as a mouse and a display cursor. Cells for holding output values of the function may be selected in the same way. Selected cells may be dragged and dropped into the function field of the function builder for definition of the selected cells as cells holding input values or output values, respectively. In a preferred embodiment, the function builder creates a new set or panel of cells to hold output function values.

The function builder may comprise components for selection by the user for specification of the function. The components may be function components, such as mathematical operators, mathematical functions, logical and Boolean functions, financial functions, string functions, data base functions, etc.

A function component may be a Java class that encapsulates the algorithm of a specific function, e.g. a mathematical function, a logical and Boolean function, a

financial function, a string function, a data base function, etc, and the data that is input for the function.

Further, the components may be data components, such as data of a set of cells, data from an external data source, etc. For example, a data component may be a Java class that holds an object reference to a specific data source, e.g. a set of cells or an external database.

Further, the function builder may comprise graphical symbols relating to respective components for selection by the user to be incorporated into the desired function. The graphical symbols relating to respective components may be organized in various ways, e.g. under various tabs according to their type, in a graphical, hierarchical diagram according to their type, etc.

The user may select a specific component to be included in the function by dragging and dropping the corresponding graphical symbols into the function field of the function builder.

The function builder may further comprise tools for specification of calculation type, such as by row, by column, all cells in a panel, selected cells in a panel, accumulated or non-accumulated, etc.

In order to be able to store and retrieve functions built with the function builder, the user interface may comprise means for naming functions built with the function builder, and may comprise means for storage and retrieval of functions in a memory.

The model builder may comprise means for storing a selected part of the model, such as the entire model, in the memory.

A first function may constitute an input variable to a second function.

The function builder may further comprise tools for user definition of a mathematical operator.

Thus, according to an embodiment of the present invention, an electronic mathematical model builder is provided having a user interface providing input tools, such as a keyboard and a pointing device, e.g. a mouse, a trackball, a pen, etc, and a display displaying a cursor that is controlled by the pointing device, to be controlled by an operator of the builder to 1) create cell panels in a model document resembling a conventional spreadsheet, 2) creating and editing functions, such as mathematical formulas, 3) assigning functions to desired cells in the model document, and 4) storing functions in a library of user defined functions.

The model builder may further include tools for saving a model document with or without data as a standalone Java .jar file or .exe file, and tools for documenting the structure of a specific model document.

With the present invention, users may construct model documents in substantially less time and with fewer errors than what is possible with conventional spreadsheet systems.

The model builder may be accessed through an Internet browser.

Different rights may be assigned to different users, e.g. so that some users may be able to display and import data from a specific mathematical model while others may be allowed to edit various parts of the model.

Further, the user may build standalone application to be executed on any Java-enabled PC, Mac, and UNIX machine or in a java-enabled browser, with no special downloads, plug-ins or spreadsheet engines required.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 illustrates the operation of a conventional spreadsheet,

Fig. 2 illustrates the operation of a conventional spreadsheet,

Fig. 3 shows a goal seek dialogue box of a conventional spreadsheet,

Fig. 4 shows a model according to the present invention as displayed on a monitor,

Fig. 5 shows various cell panels,

Fig. 6 shows a conventional spreadsheet display,

Fig. 7 shows a hidden formula of the spreadsheet of Fig. 6,

Fig. 8 shows other hidden formulas of the spreadsheet of Fig. 6,

Fig. 9 shows the example of Fig. 6 as modelled according to the present invention,

Fig. 10 shows function builder displays,

Fig. 11 schematically illustrates the architecture of an embodiment of the present invention,

Fig. 12 schematically illustrates interfaces and components of an embodiment of the present invention,

Fig. 13 shows an Excel spreadsheet,

Fig. 14 shows hidden formulas of the spreadsheet in Fig. 13,

Fig. 15 shows a function builder display example according to the present invention,

Fig. 16 schematically illustrates a relation between a cell panel and a function,

Fig. 17 shows another function builder display example according to the present invention,

Fig. 18 shows yet another function builder display example according to the present invention and the architecture of the function,

Fig. 19 schematically illustrates JAVA building blocks of an embodiment of the present invention,

Fig. 20 schematically illustrates different layers of an embodiment of the present invention,

Fig. 21 schematically illustrates JAVA classes of an embodiment of the present invention,

Fig. 22 schematically illustrates JAVA objects of an embodiment of the present invention,

Fig. 23 shows a function builder display according to the present invention,

Fig. 24 shows a flow chart of calculation of an output value of a function according to the present invention,

Fig. 25 shows a function created with the function builder according to the present invention,

Figs. 26 – 34 illustrates the process of creating a function with the function builder according to the present invention,

Fig. 35 shows the function builder displaying a retrieved function,

Fig. 36 is a blocked schematic of an embodiment of the present invention, and

Fig. 37 is a perspective view of the electronic mathematical model builder embodiment of Fig. 36.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

The description in this paper relies on technical terms used in the subjects of object-oriented programming and specially Java programming. For an overview, see e.g., Ken Arnold, James Gosling & David Holmes, The Java Programming Language Addison-Wesley 2000, 3'rd edition ISBN 0-201-70433-1

The embodiment illustrated in the appended figures is programmed in the programming language Java and the system is running on the Java platform. Other programming languages could have been used for the development (e.g. C++ or C#) for a specific platform (e.g. Windows XP or Mac OS).

As shown in Fig. 4, the illustrated embodiment of the present invention includes a spreadsheet system having a user interface built of Java graphical user interface components (e.g. windows, drop down menus (4.1), toolbars (4.2), buttons, tabbed panes (4.3), labels, scroll bars etc. (4.4)). In operation, the system provides a number of cell-groups (grid views) organised in panels (Fig. 5a). The system provides means for user input into the cells (e.g., keyboard, mouse, disk, and the like) for storage and processing by the system. The system provides means for building and holding spreadsheet formulas and means for recalculating of formulas and showing the result in the system (4.7). Further, the system provides means for drawing and updating charts (4.6).

In the illustrated embodiment of the present invention, data are stored in cells, which in turn are grouped in panels (Fig. 4.5). A panel can hold one cell, a column of cells, a row of cells or a two-dimensional or a N-dimensional ($N \geq 3$) matrix of cells. A cell panel have row labels, column labels and a unique identifier (in the illustrated embodiment of the present invention, a name in the embodiment of a Java String). A cell panel has further a number of attributes to describe the state of the panel (e.g. the visibility of the row and column labels, the colour of the labels and cells, edibility of the cells, number format etc.).

In the illustrated embodiment of the present invention, the number of cells in a cell panel can be greater than the number of visible cells. The system provides means for the user to adjust the number of cells in a cell panel as well as the number of visible cells. Further, the system provides means for attaching a scrollbar to one or a group of cell panels with the purpose of scrolling cells not visible into the visible area of the cell panel. In Fig. 5a and Fig. 5b it is shown, that the scrollbar in the cell panel Column (5a.1) scroll the cells in cell panel Column as well as in cell panel Column 2 (5a.2 and 5b.4)

In the illustrated embodiment of the present invention, a user creates a cell panel by dragging a cell tool (4.2) from a toolbar and dropping it in the work area (4.8) (the spreadsheet document). The user can edit the attributes of a cell panel in a dialog box and change the location of a panel by moving it with a hardware device (e.g. a mouse or a keyboard).

According to the present state of the art, development of spreadsheet models involves working with text based pr. cell formulas either by typing the formula directly in the cells or by using a formula builder, for building text based spreadsheet formulas (for a description se e.g. United States Patent 5,603,021). The formulas and raw data (e.g. numeric values) are mixed together in the user interface

In Fig. 6 it is shown how data and formulas are mixed together in the cells in a spreadsheet model to obtain a model calculation prices inclusive VAT. Many empty cells in the spreadsheet are provided, but not needed for the spreadsheet model.

In Fig. 7 it is shown, that cell C1 has a text-based formula entered in the cell. In fact the cells C1 to C12 all have text-based formulas of almost the same appearance, as shown in Fig. 8.

Fig. 9 shows the same model in the illustrated embodiment of the invention. MAT uses drag and drop techniques to create the number of cells needed for the calculation (25 cells). The cells are grouped in cell panels.

In the illustrated embodiment of the present invention, formulas are not text based but built of software components. A formula can be aggregated by other formula components. To build a formula MAT uses a graphical builder based on drag and drop techniques. Further, a formula in MAT is not associated with a single cell but a complete cell panel. This means that many cells can share the same formula.

In Fig. 10 it is shown that the model has only one formula, which is associated to the cell panel with the column label "Input"VAT" (10.4). All cells in the Cell panel are properly calculated, the type of calculation is performed row by row (10.1) and the cell panel "Vat" provides data as absolute references (10.3)

One of the fundamental problems in the art of spreadsheets is that data and functionality are mixed together in the user interface (7.1).

To overcome this problem according to the present invention, the illustrated embodiment uses an extended MVC (Model View Controller) architecture shown in Fig. 11.

Fig. 11 shows the interaction between the components in the MVC architecture. The Grid Model (11.2) passes data to the Cell (11.1). The Grid View determines which events that are passed to the Grid Controller (11.3), e.g. key events, when the user has entered new data into the Grid View or mouse events. The Grid Controller connects the Grid Model with the Grid View and determines how the Grid Model shall react on a specific event.

The Grid Model does not encapsulate the data as in a traditional MVC architecture. Instead the Grid Model has an association to an object of type Data Layer that encompasses the data (11.4). The Data Layer is an abstract class and has in the illustrated embodiment of the present invention four non-abstract subclasses (11.5).

By provision of this extension to the MVC architecture, the data have been separated from the user interface, and during the builder process, it is possible to work interactively with the functionality e.g. changing the kind of data or the type of calculation without disturbing the user interface.

In the illustrated embodiment of the present invention, Data Layer is an abstract class that encapsulates all fields and methods that are common for the specialized Data Layers: Integer Layer, Double Layer, Object Layer, Calc Layer (11.6) etc.

The Double Layer, for instance holds primitive data of type double while the Object Layer can hold mixed data (e.g. data of type String and data of type Integer).

The Calc Layer holds the results of recalculations and will be investigated further in the next couple of sections.

As schematically illustrated in Fig. 12, MAT defines three interfaces that manage the interrelationships between the components, Data Provider (12.1), Cell Provider (12.3) and Data Observer (12.4).

The two interfaces Data Provider (12.1) and Cell Provider (12.3) specify the behaviour for components that can have dependent components (e.g. a cell panel can provide input for a chart. Whenever the data in the cell panel changes, the chart needs to be repainted). A Data Provider is used when a complete cell panel provides data and a Cell Provider is used when only part (e.g. a single cell or a group of cells) of a cell panel provides the data.

In the illustrated embodiment of the present invention, components that can have dependent components must implement one of these interfaces.

Components that are dependent on other components (e.g. can be recalculated or repainted) must be of type Data Observer (12.4). The illustrated embodiment defines the interface Data Observer to specify behaviour for components that can react on changes in components of type Data Provider.

Both the Calc (12.6) component and the Chart (12.5) component implements the Data Observer interface.

Spreadsheet formulas are fundamental to the creation and operation of a spreadsheet data model. In the state of the art, during creation of a particular traditional spreadsheet or worksheet model, a user enters formulas as text in spreadsheet cells the same way he or she enters values and labels.

A spreadsheet formula has a specific syntax defined by the spreadsheet producer. A formula begins with a special token (e.g. = in Excel) to distinguish the formula from raw data, followed by tokens as cell references (e.g. C1), operators (e.g. + - /), values, and built in functions (e.g. SUM(C1:C12)). Required arguments to a built-in function must be enclosed in parentheses. Multiple arguments must be separated by a semicolon, or other argument separator (as specified by the implementation). Further, multiple arguments must be entered in a specific order. If an optional argument is specified for a built-in -function, the user must also specify all preceding optional arguments for the function. If a user fails to follow the specified format for entering a built-in function formula, a spreadsheet system will refuse to process the formula further and, instead, display a syntax error message.

In general, users must adhere to the following prescribed format for functions:

FUNCTION_NAME (Argument 1; Argument; . . .)

The type of information to be provided as arguments depends on the specific built-in function. Arguments can be divided into three general types: numeric values, matrix values, and string values (text). Most arguments require one of these types of values. Some accept a combination or a choice of types. For instance, SUM and other statistical functions accept a "list," which can be a matrix value (i.e., cell coordinates e.g.: C1:C20) used alone or with numeric values.

Users can use more than one built-in function in a single formula, with one built-in function serving as the argument or parameter to another built-in function. This is called "nesting" functions. Generally, a user can nest as many functions as desired, several layers deep. For example, the following calculates the average values in cell matrix C14 . . . F24, then rounds the average to two decimal places:

=ROUND (AVG (C14: F24), 2)

Nested functions evaluate from inside out. In the above example, for instance, the AVG is calculated before the ROUND.

When formula syntax and arguments are correct, most built-in functions and formulas evaluate to a numeric value (including dates) or string value (label or text). If a value

cannot be calculated, a spreadsheet "ERR" value is returned thereby indicating an error condition.

In Fig. 13 a typical Excel spreadsheet model is shown. The cells A1 to A12 and B1..B12 hold some data, while the cells C1..C12 hold formulas.

In the state of the art the evaluation of formulas (shown in Fig. 14) will not be based directly on the formulas entered by the user, but on a "Reverse-Polish Notation" (RPN) or "postfix" format. Before recalculation the spreadsheet system converts the formulas to (RPN) format: E.g. the formula A1+B1/5 entered by the user as text is converted to

A1 B1 5 / + (13)

As the RPN indicates, the / operator has higher precedence than the + operator. According to this the term B1/5 will be evaluated first and the result of this division will be added to value hold in cell A1.

The RPN format will make it possible for a traditional spreadsheet system to carry out a stack based recalculation. The internal representation of the formula in RPN format may be in so called "p-code" or opcodes, easy to convert back into text strings ("de-parsed") for easy on-screen editing of formulas by the user.

In general in the state of the art, a spreadsheet system holds a so called spreadsheet engine able to 1) parse a spreadsheet formula entered by the user as text to the internal representation 2) de-parse the internal representation back into a text string readable for the user 3) sort the cell dependency into a specific recalculation order (e.g. natural recalculation order) and 4) evaluate the formula to a numeric value (including dates) or string value (label or text) to be shown in the spreadsheet at the specific cell location. If a value cannot be calculated, a spreadsheet "ERR" value is returned thereby indicating an error condition.

In MAT a spreadsheet formula is constructed of software building blocks, which resemble the LEGO building blocks for children. In MAT a spreadsheet formula can consists of one or many software-building blocs. A spreadsheet formula can be constructed by software components in much the same manner children builds houses of LEGO blocks by placing building blocks side by side or on top of each other. A software building block represents 1) a function (e.g. a sum function) - then called a software function building block - able to perform calculation on data input or 2) fundamental data (e.g. holding a reference to a cell panel or a reference to a constant value) – then called a software data building block.

In MAT it is further possible to embed one or many software building blocks into software function building blocks and thereby nest the building blocks. Nested building blocks can be software function building blocks and/or software data building blocks. The outer building block then acts as a container for the nested building block(s). The nested building block(s) makes up input data for the container building block.

After the construction of a formula in MAT. the formula is 1) associated with a cell panel, which then shows the result of the calculation and/or 2) stored in a library of user-defined formulas.

A formula in MAT can consist of a function, e.g. a Sum function, which sums all the cells in a cell panel (Fig. 15a.1) C1. The formula is constructed by the software function building block (15a.2) representing a sum function. Embedded in the function is a software data building block with a reference to the cell panel C1. The formula is associated to the cell panel (15a.3), which shows the result of the calculation. Further the formula sums all cells in the embedded software data building block and shows the result in one cell. The type of calculation is specified to calc all (15a.4). Calculation types will be specified below.

Fig. 16 shows the software function building block representing a sum function with an association to the software data building block representing the cell panel C1.

A software function building block can further have embedded software function building blocks as shown in the next example (Fig. 17). Two cell panels C1 and C2 hold input data and a cell panel Sum holds output data generated by the formula.

The formula (17.4) is built of a software function building block representing a Sum function and two embedded software building blocks. The first is a software data building block having a reference to the cell panel C1 (17.1) and the second is a software function building block representing a division function. The division software function building block has two embedded software building blocks. The first is a software data building block having a reference to the cell panel C2 (17.2). The second is a software data building block having a reference to a const value 5.0.

The formula (17.4) is performing the calculation in rows as specified by the dropdown menu (17.6). The meaning of this is that the calculation is performed row by row starting from the top (e.g. the value in the first cell in C1 is added to the value in the first cell in C2 divided by 5 (1 + 1/5 = 1.2 and the result is placed in the first cell in the Sum cell panel) then the second cell in C1 is added to the value in the second cell in

C2 divided by 5 (2 + 2/5 = 2.4 and the result is placed in the second cell in the Sum cell panel etc.)

Fig. 18a and 18b shows the architecture of the formula. The Sum software function building block (18b.1) has two embedded software buildings blocks (18b.2). The second of these buildings blocks is another software function building block with two embedded software building blocks (18b.3).

In one embodiment of the present invention, a Java interface Function (19.1) is provided, as illustrated in Fig. 19, to define the behaviour for calculation in the system. Further an extendable abstract Java class Function (19.2) is provided. The class implements the interface Function and defines the behaviour for built-in functions in the system. All built-in functions (19.3) in the illustrated embodiment extend the abstract class Function.

The system provides further a Java class Function Holder (19 4), able to contain/hold software building blocks. The class further has an object reference to an object of type Function. Objects of type Function Holder can thus act as a built-in function and carry out calculations.

All spreadsheet formulas in the illustrated embodiment of the present invention are of type Function.

After the construction of a formula in MAT, the formula is associated with a cell panel, which then shows the result of the calculation and/or stored in a library of user-defined formulas.

In the illustrated embodiment of the present invention, an association of a formula to a cell panel is done by utilization of the extended MVC architecture. In Fig. 11 the association (object reference) from the Grid Model (11.2) to the Data Layer (11.4) is set to type Calc.

As schematically illustrated in Fig. 20, the Calc layer is a special Data Layer having an association (object reference) (20.2) to an object of type Function. By this object reference the calc layer holds a formula and through the mechanism of the extended MVC architecture the result of a calculation is shown in the cell panel.

The type of calculation performed by a specific formula is specified by the calc layer and implemented as an association (object reference) (20.3) to an object of type Calc Type (21.1) as schematically illustrated in Fig. 21. In the illustrated embodiment of the present invention, the Calc Type is a Java interface. The system defines a number of Java classes that implements the Calc Type interface (21.2).

In the illustrated embodiment of the present invention, each function can have an associated private calc type specified by an object reference from an object of type Function (22.1) to an object of type Calc Type (22.2). If no special Calc Type is provided for a function the object reference (22.1) is set to null, and the Calc Type of the function will be the same as specified by the Calc Layer. The default Calc Type for the Calc Layer is "in rows" and cannot be null. The Calc Type for a function has higher precedence than the Calc Type for the layer.

In the illustrated embodiment of the present invention, a formula built of a number of functions can in the process of calculation carry out different calculations types. Fig. 23 shows an example.

The cell panel PRO4 (23.3) displays the result of the calculation of the formula (23.4). The formula combines three functions: 1) Product 2) Sum and 3) Avg. The calc type associated with the calc layer is calc row (calc in rows). To the Sum function there is further assigned a private calc type Calc All (23.6). As the Product and the Avg functions have no assigned private Calc Type, the calculation of these functions will be in rows, specified for the Calc Layer (23.5).

The value (15) in the first cell in the cell panel PRO 4 (23.3) is a result of the calculation Product (10; 1.5) that evaluates to 15. The first argument, 10 is a result of the calculation Sum (Column 1) as the calc type of the Sum function is Calc All (23.6) and all the cells in Column 1 is added to one value. The second argument 1.5 is a result of the calculation Avg (the value in the first cell of Column 2 ; the value in the first cell of Column 2) equal to Avg (1;2).

The value (20) in the second cell in the cell panel PRO 4 (23.3) is a result of the calculation Product (10; 2) that evaluates to 20. The first argument 10 is a result of the calculation Sum (Column 1) as described above. The second argument 2 is a result of the calculation Avg (the value in the second cell of Column 2 ; the value in the second cell of Column 3) equal to Avg (2;2). Etc.

The evaluation of a formula, in the illustrated embodiment of the present invention, is a simple traverse of the arguments left to right. The user will evaluate the formula directly on the formula built. The formula does not need to be converted to a RPN notation before evaluation.

A Formula evaluation, in the illustrated embodiment of the present invention, is done by a procedure called "Handle Function." Fig. 24 shows a flow chart of formula evaluation of the illustrated embodiment of the present invention:

Let us follow the flow chart (Fig. 24) to evaluate the formula shown in Fig. 25. The formula is built of a software function building block representing a Sum function (Fig. 25.1) with two embedded software building blocks (25.2) and (25.3).

The evaluation of the formula starts by making a call to the procedure Handle Function in (24.1). Let us name this procedure call, the outer procedure. The task for this procedure is to evaluate the outer software function building block, representing the Sum function (Fig. 25.1). The evaluation moves to (24.2). Since the software function building block has two embedded software building blocks the decision in (24.2) is set to true and the evaluation moves to (24.3). At this point the procedure gets the next software building block (C1 in Fig. 25.2) and moves to (24.4). As the software building block (Fig. 25.2) is a software data building block the decision in (24.4) is set to false and the data building block is added to the list of operands (24.5). The evaluation proceeds to 24.2. As one more embedded software-building block is present (Fig. 25.3) the evaluation get hold of this building block (24.3) and moves to (24.4). The decision in (24.4) is set to true, as this building block is a software function building block. The evaluation then moves to (24.6). At this point a new call to the procedure Handle Function is made. Let us name this procedure call, the inner procedure. The task for this inner procedure is to evaluate the inner software function building block, representing the Division function (Fig. 25.3). The inner procedure starts in (24.1) and moves to (24.2) in order to check the number of embedded software building blocks. As the software function building block has two embedded software building blocks (Fig. 25.4 and 25.5) the decision in (24.2) is set to true and the evaluation moves to (24.3) where it gets the next software building block (25.4). The evaluation moves to (24.4) in order to determine the type of the actual software building block. The type of the software building block is a software data building block (Fig. 25.4) and the decision in (24.4) is set to false and the data building block (C2 in Fig. 25.4) is added to the list of operands (24.5). The evaluation proceeds to 24.2. Since one more embedded software block is present the evaluation gets the next building block (24.3) and moves to (24.4). The new software building block is also a software data building block and is added to the list of operands and the evaluation moves to (24.2). As no more embedded software building blocks are present the evaluation moves to (24.7). The software function building block executes the function it represents (Division) on the operands which is found in the list of operands and the result of the inner Handle Function procedure is returned to the outer Handle Function procedure. The result is returned in (24.5) of the outer procedure and added to the list of operands.

The outer procedure continues the evaluation and moves to (24.2). No more embedded software building blocks are present and the evaluation moves to (24.7). The software function building block executes the function it represents (Sum) on its operands and the result is returned. As the software function building block representing the Sum function is the outer software building block the result of the evaluation is returned to the cell panel, it is associated with (e.g. Sum in Fig. 18).

The work process of creating a spreadsheet model (e.g. a financial model) typical involves setting up data and building formulas, re-building formulas, changing the position of the data and formulas, setting up charts etc.

In the illustrated embodiment of the present invention, a formula is built of software building blocks included in the actual model by drag and drop techniques.

To be able to build formulas, the illustrated embodiment of the present invention provides a visual tool for building formulas (Fig. 26) Formula Builder. In the illustrated embodiment, the Formula Builder is a javax.swing dialog (26.2), which is opened by a click on the Function tool in the tools palette (26.1).

The Formula Builder includes a Tabbed Pane with a number of tabs to organise the different types of built-in functions (e.g. Statistical and Logical) (26.3). On each tab pane the software function building blocks that represent the built-in functions are placed (26.4) able to be dragged into the work area of the Formula Builder (26.8). The Formula Builder further includes a tool (26.5) to specify the calculation type of the formula, (26.6) a tool to associate a formula with a cell panel and (26.7) a tool to store a formula in a library.

To build a new formula the user drops a function into the work panel (Fig. 27.1 and 27.2)

The user can embed software building blocks into a function. With a mouse click (or another input device e.g. a keyboard) a new dialog box is opened to represent the internal of the function (28.2). In this "inner" dialog box most of the same tools are available as in the "outer" dialog box. A work area is provided (28.2) into where the user can drag software building blocks. In the illustrated embodiment, two extra tab panes are provided (28.3 and 28.4). In the tab pane "Const" a software data building block is provided to add constant values to a formula (28.5).

In the tab pane "Column" a software data building block is provided to add software data building blocks that represent cell panels in the spreadsheet model to a formula (29.2).

In Fig. 30 the complete formula is displayed. The "outer" software function building block representing an "If" built-in function (30.1) has three embedded software building blocks as schematically illustrated in Fig. 31 window 2. A software function building block representing an Equal To built-in function (30.2) and two software data building blocks (30.3 and 30.4). The "inner" software function building block representing an Equal To built-in function (30.2) has two embedded software data building blocks as illustrated in Fig. 31 window 3.

The user can associate the formula with a cell panel in the spreadsheet model by clicking on the button 5 in Fig. 30 or the user can store the formula in a library of user-defined formulas by clicking the button 6 in Fig. 30.

In Fig. 32a the cell panel associated with the formula is shown (32a.1). In the illustrated embodiment of the present invention, the user can open the Formula Builder by double clicking on the cell panel (32b.1). In this way the user can be built the formula.

As an example the user can reorder the embedded software building blocks in the formula, cf. Fig. 33a, item 1 by dragging and dropping the software building blocks. The result is shown in Fig. 33b, item 1.

In the illustrated embodiment of the present invention, a user can store a user-defined formula in a library. The user may associate a formula with a cell panel by clicking a button, Fig. 30, button 5. By clicking button 6 in Fig. 30, the same formula could be stored in a library to be used later in the same spreadsheet library or in other spreadsheet models. To be able to identify the new "built-in" formula, the user will be asked to give it a name, cf. Fig. 34, window 2 (e.g. IF_EqualTo). The formula will then appear in the Formula Builder as a user defined built-in formula, cf. Fig. 35, window 1. The formula can be dropped into the work area of the Formula Builder, cf. Fig. 35, window 2, and software building blocks can be embedded, cf. Fig. 35, window 3. In Fig. 35, the formula takes four embedded building blocks equivalent to the four software data building blocks in Fig. 30.

Fig. 36 is a blocked schematic of the embodiment shown in Fig. 37. The illustrated electronic mathematical model builder 10 according to the present invention, includes a central processing unit (CPU) 14 with a math coprocessor or floating-point processing unit (FPU), a main memory 20, an input/output controller 24, a keyboard 16, a pointing device 18, e.g., mouse, track ball, pen device, or the like, a display device 12, and a mass storage 22 (e.g., hard disk). Additional input/output devices, such as a printing device 26, may be included in the system 10 as desired. As

illustrated, the various components of the system 10 communicate through a system bus 28 or similar architecture.

A computer software system is provided for directing the operation of the computer system 10. The software system is stored in system memory 20 and on disk memory 22, includes a kernel or operating system and an interface. One or more application programs may be "loaded" (i.e., transferred from storage 22 into memory 20) for execution by the system 10. The system 10 receives user commands and data through the user interface, i.e. the keyboard 16 and the pointing device 18; these inputs may then be acted upon by the system 10 in accordance with instructions from operating system module and/or the application module. The interface, which may support a character-based and/or preferably a graphical user interface (GUI), also serves to display results, whereupon the user may supply additional inputs or terminate the session.

Fig. 36 is a perspective view of an electronic mathematical model builder 10 according to the present invention. The illustrated embodiment 10 includes a personal computer with a processor 14, a memory 20, an internal hard disc for mass storage 22, a CD-ROM drive, an audio capability, a display 12, a keyboard 16, and a pointing device, such as a track ball or a mouse 18. In Fig. 36, the image of Fig. 4 is shown on the display screen. The display 12 may have a touch sensitive display screen. It will be appreciated by those skilled in the art that the actual hardware implementation of the system is a matter of design choice. Thus, the model builder may be distributed across a network connected to a server, e.g. it may be provided by an on-line service for example via the Internet. For example, the computer 10 may be a portable computer that can be connected to the Internet via a mobile phone or via a wireless network, e.g. a hot spot, or the user may be connected to the Internet via a broadband connection, e.g. utilising the user's TV.

CLAIMS

1. An electronic mathematical model builder comprising

a memory for storage of data,

a processor for defining addressable sets of cells stored in the memory with a unique identifier, for entering data into the cells and for processing data stored in the cells,

a user interface with a display for displaying sets of cells in a work area and means for creating and positioning sets of cells in the work area and means for specifying data to be entered into the cells, and

a function builder for building mathematical relations between cells, comprising fields for user specification of a desired function by mathematical operators and input variables of the function, and a set of destination cells for containing values of the function.

2. A model builder according to claim 1, wherein the function builder comprises components for selection by the user for specification of the function.

3. A model builder according to claim 2, wherein at least one component is a function component selected from the group consisting of mathematical operators, mathematical functions, logical and Boolean functions, financial functions, string functions, and data base functions.

4. A model builder according to claim 3, wherein the function component is a Java class that encapsulates the calculation of a specific function, e.g. a financial or mathematical function, and the data that is input for the function.

5. A model builder according to any of claims 2-4, wherein at least one component is a data component including data of a set of cells.

6. A model builder according to claim 5, wherein the data component is a Java class that holds an object reference to a specific data source, e.g. a set of cells or an external database.

7. A model builder according to any of the preceding claims, further comprising data components including data from an external data source.

8. A model builder according to any of the preceding claims, wherein the function builder comprises graphical symbols relating to respective components for selection by the user to be incorporated into the desired function.

9. A model builder according to claim 8, wherein the graphical symbols relating to respective components are organized under various tabs according to their type.

10. A model builder according to claim 8, wherein the graphical symbols relating to respective components are organized in a graphical, hierarchical diagram according to their type.

11. A model builder according to any of claims 8-10, wherein components are selected by dragging and dropping the corresponding graphical symbols into the function field of the function builder.

12. A model builder according to any of the preceding claims, wherein the function builder comprises tools for specification of calculation type, such as by row, by column, all cells in a panel, selected cells in a panel, accumulated or non-accumulated.

13. A model builder according to any of the preceding claims, wherein the user interface comprises means for naming functions built with the function builder.

14. A model builder according to claim 13, further comprising means for storage and retrieval of functions in the memory.

15. A model builder according to any of the preceding claims, further comprising means for storing a selected part of the model in the memory.

16. A model builder according to any of the preceding claims, wherein a first function may be an input variable to a second function.

17. A model builder according to any of the preceding claims, wherein the function builder further comprises tools for user definition of a mathematical operator.

18. A model builder according to any of the preceding claims, further including tools for saving a model document as a standalone Java .jar file.

19. A model builder according to any of the preceding claims, further including tools for saving a model document as a standalone Java .exe file.

20. A model builder according to any of the preceding claims, further including tools for documenting the structure of a specific model document.

Cell	Formula	Comments
A4	=B1*A1	VAT
B4	=IF(A4 <= 100;A4*C1;A4*C1*1.5)	Shipping cost
A7	= A1+A4+B4	Total cost

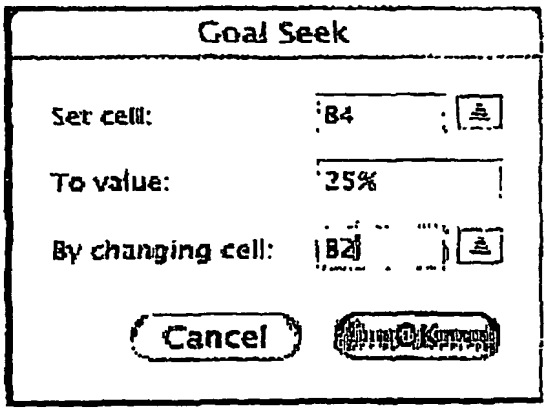
(Prior art)

Fig. 1

Cell	Formula	Comments
B2	8500	sales
B3	6800	costs
B4	=B2-B3	gross profit
B5	=B4/B2	gross profit percentage

(Prior art)

Fig. 2



(Prior art)

Fig. 3

C

=A1*B$1
=A2*B$1
=A3*B$1
=A4*B$1
=A5*B$1
=A6*B$1
=A7*B$1
=A8*B$1
=A9*B$1
=A10*B$1
=A11*B$1
=A12*B$1

(Prior art)

Fig. 8

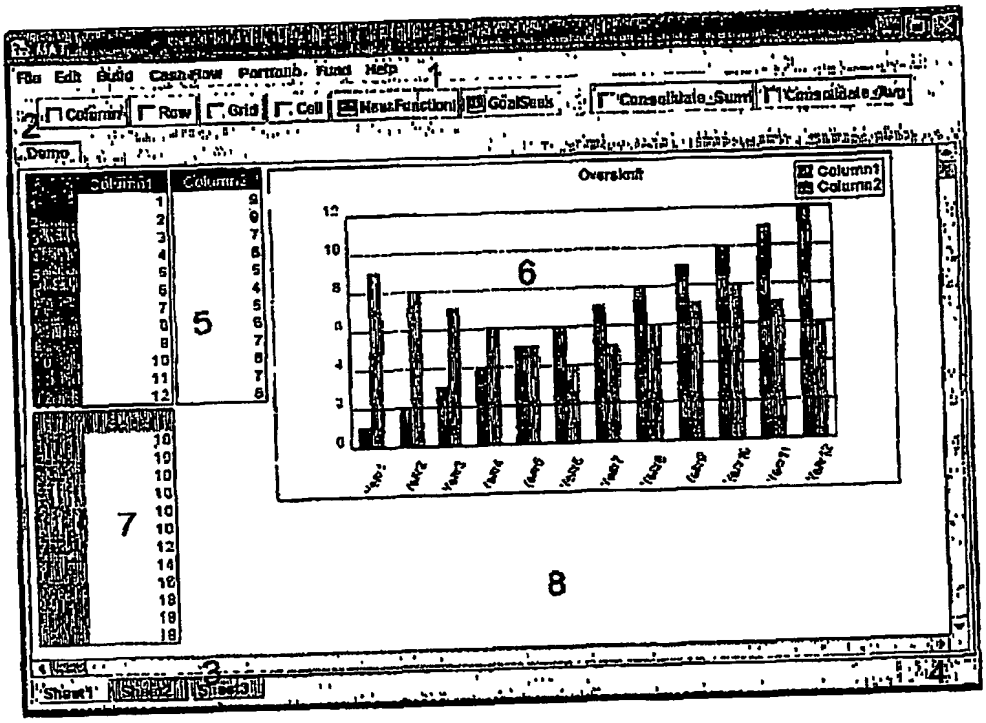

File Edit Data CashFlow Formats Find Help
Column
Row
Grid
Cell
NewFunction
GoalSeek
Consolidate Sum
Consolidate Avg
Demo
Column1
Column2
Overskrift
5
6
7
8
Sheet1
Sheet2
Sheet3

Fig. 4

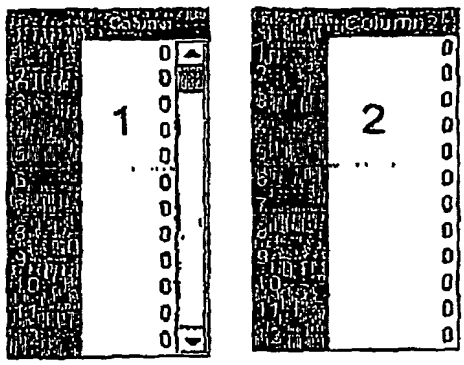


Fig. 5a

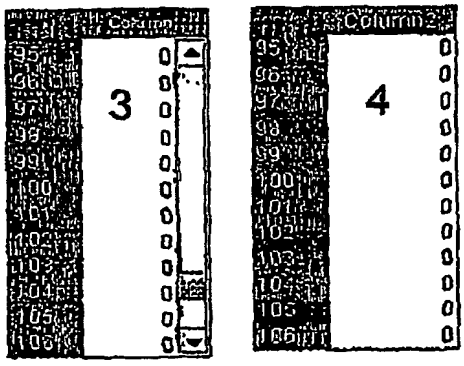
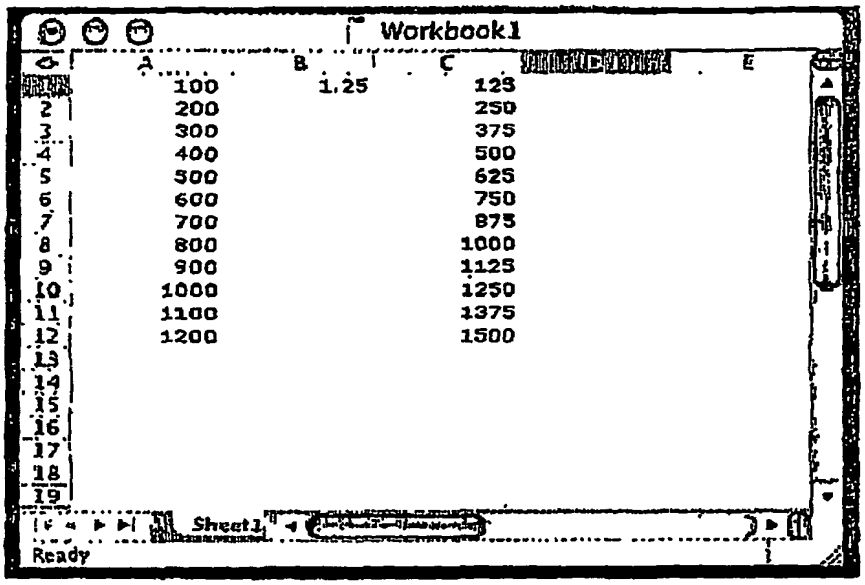
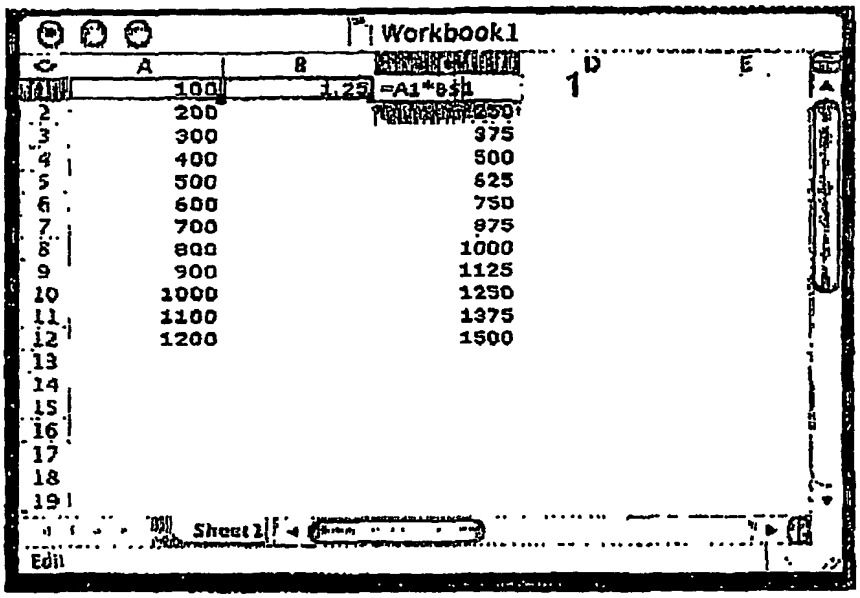


Fig. 5b



(Prior art)

Fig. 6



(Prior art)

Fig. 7

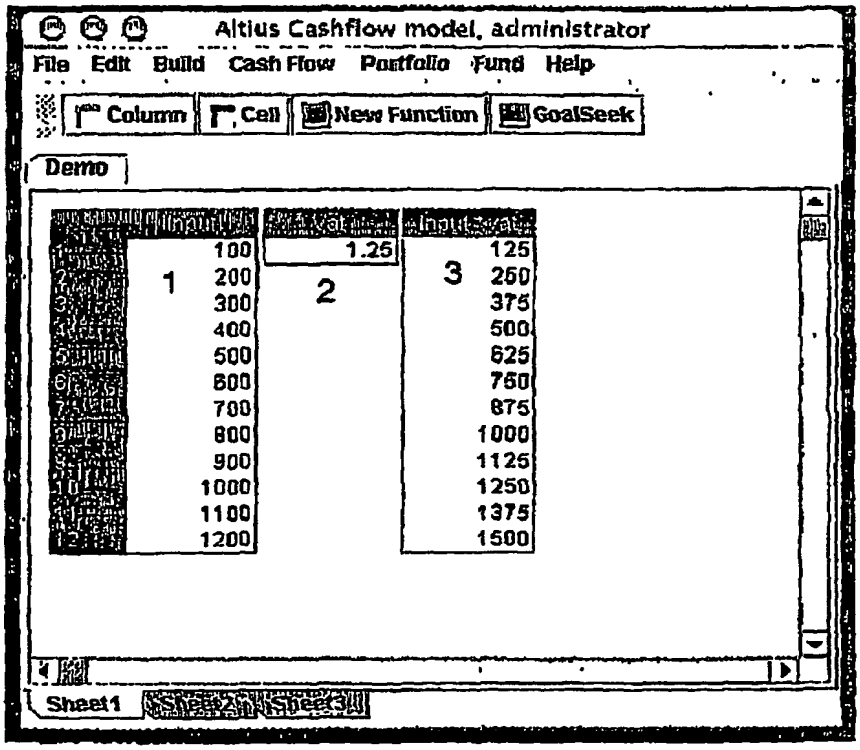

Altius Cashflow model, administrator
File Edit Build Cash Flow Portfolio Fund Help
Column
Cell
New Function
GoalSeek
Demo
Input
Input
100
200
300
400
500
600
700
800
900
1000
1100
1200
1.25
125
250
375
500
625
750
875
1000
1125
1250
1375
1500
1
2
3
Sheet1

Fig. 9

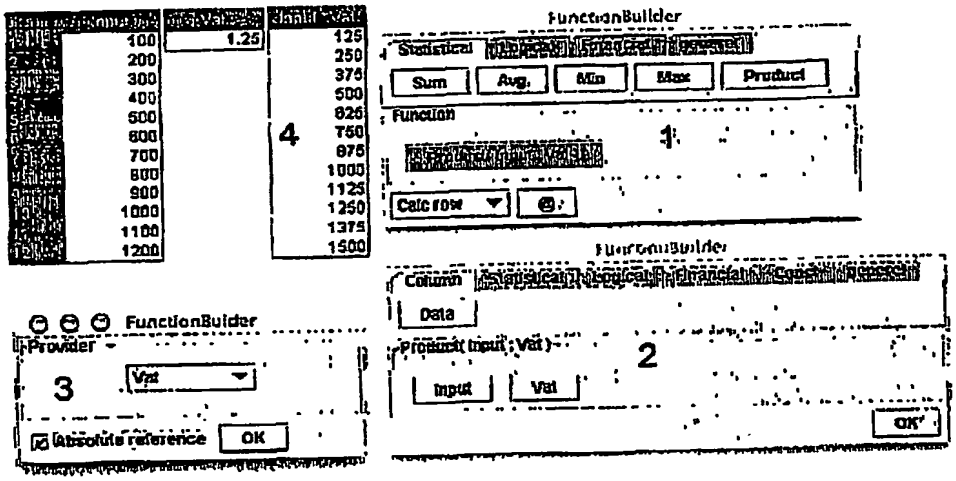

100
200
300
400
500
600
700
800
900
1000
1100
1200
1.25
125
250
375
500
625
750
875
1000
1125
1250
1375
1500
4
FunctionBuilder
Statistical
Sum
Avg.
Min
Max
Product
Function
1
Calc row
FunctionBuilder
Column
Data
Input
Vat
2
OK
FunctionBuilder
Provider
3
Vat
Absolute reference
OK

Fig. 10

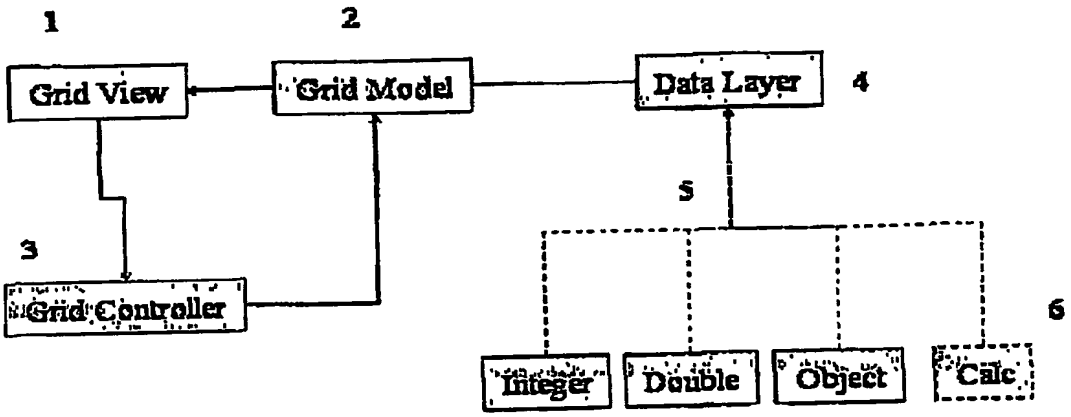

1
2
Grid View
Grid Model
Data Layer
4
5
3
6
Grid Controller
Integer
Double
Object
Calc

Fig. 11

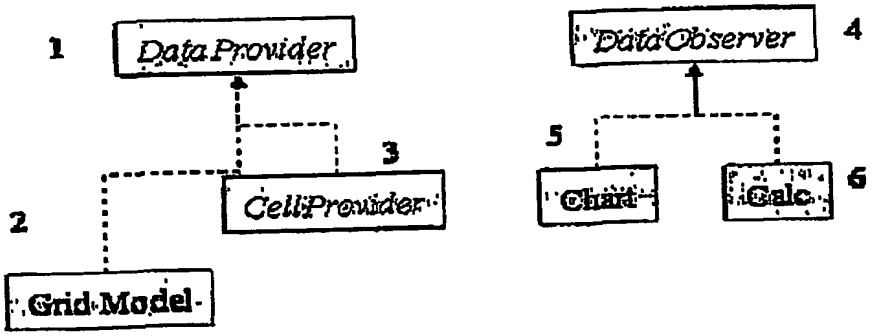

1
Data Provider
Data Observer
4
5
3
Cell Provider
Chart
Calc
6
2
Grid Model

Fig. 12

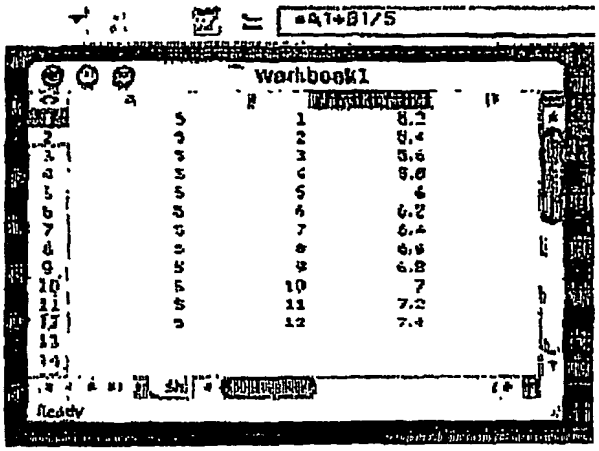


Fig. 13

	C
1	=A1+B1/5
2	=A2+B2/5
3	=A3+B3/5
4	=A4+B4/5
5	=A5+B5/5
6	=A6+B6/5
7	=A7+B7/5
8	=A8+B8/5
9	=A9+B9/5
10	=A10+B10/5
11	=A11+B11/5
12	=A12+B12/5

Fig. 14

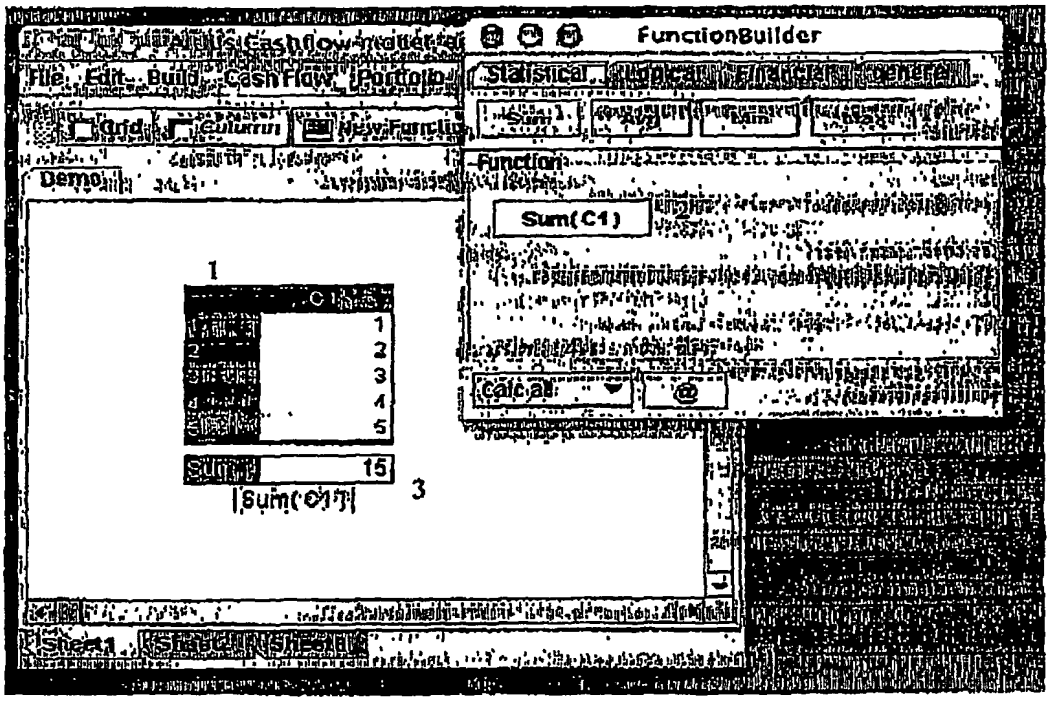

FunctionBuilder
File Edit Build CashFlow Portfolio
Statistical
Function
Sum(C1)
Demo
1
2
3
4
5
15
3

Fig. 15

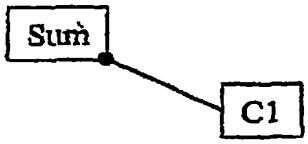

Sum
C1

Fig. 16

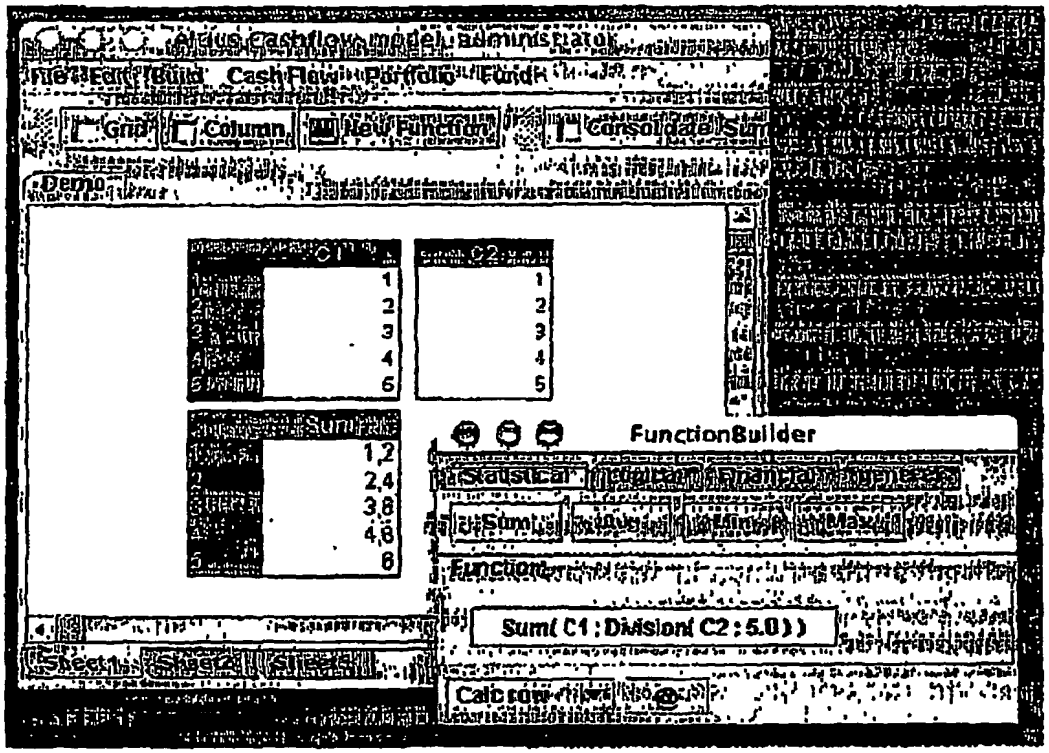

Grid
Column
New Function
Consolidate
Demo
C1
C2
Sum
FunctionBuilder
Statistical
Function
Sum(C1 ; Division(C2 ; 5.0))
Sheet1
Sheet2

Fig. 17

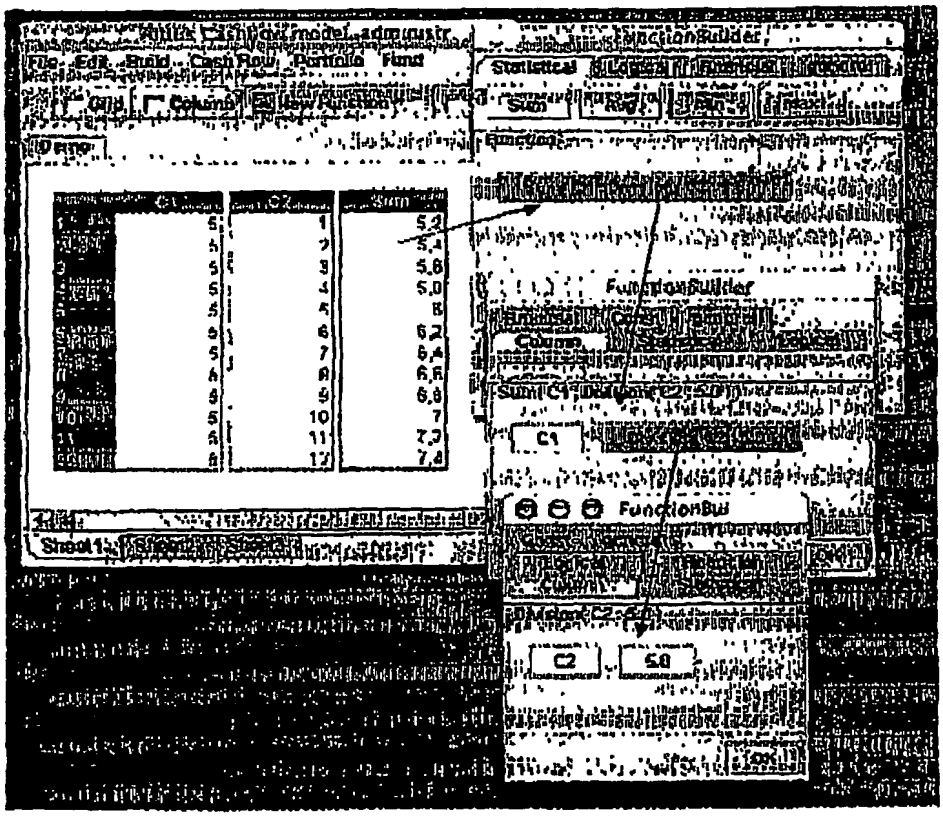

File Edit Build Cash Flow Portfolio Funct
Statistics
Sum
FunctionBuilder
Column
C1
C2
S0
Sheet1

Fig. 18a

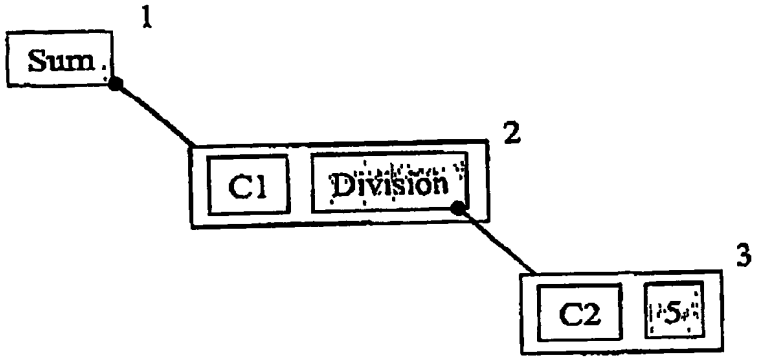


Fig. 18b

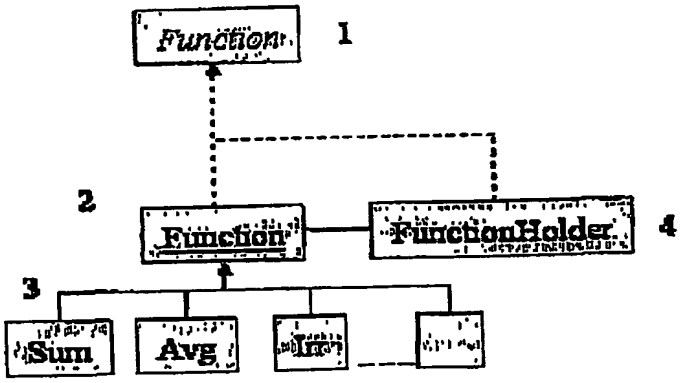


Fig. 19

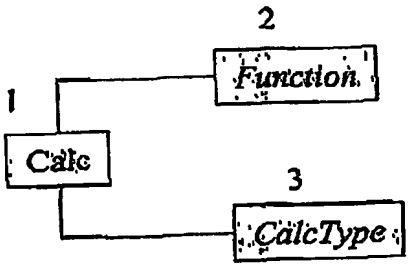

2
Function
1
Calc
3
CalcType

Fig. 20

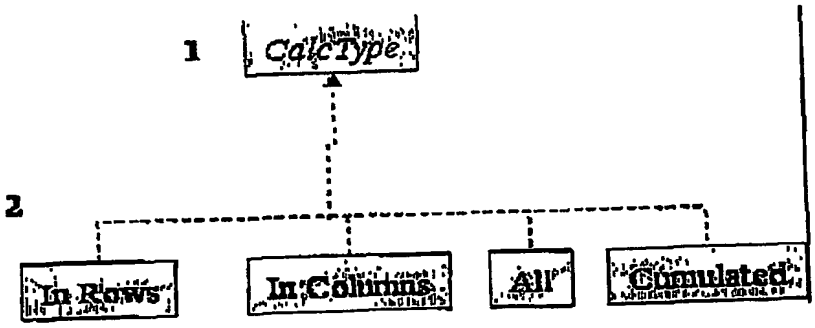

1
CalcType
2
In Rows
In Columns
All
Cumulated

Fig. 21

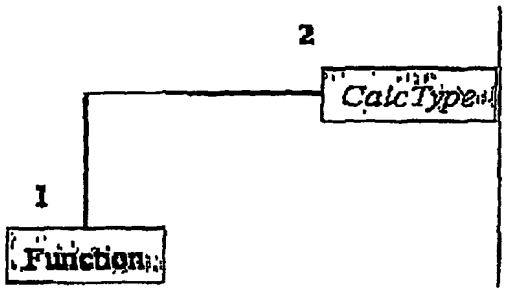


Fig. 22

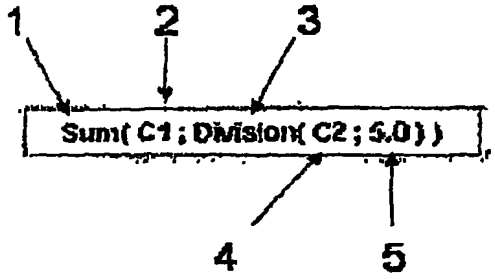


Fig. 25

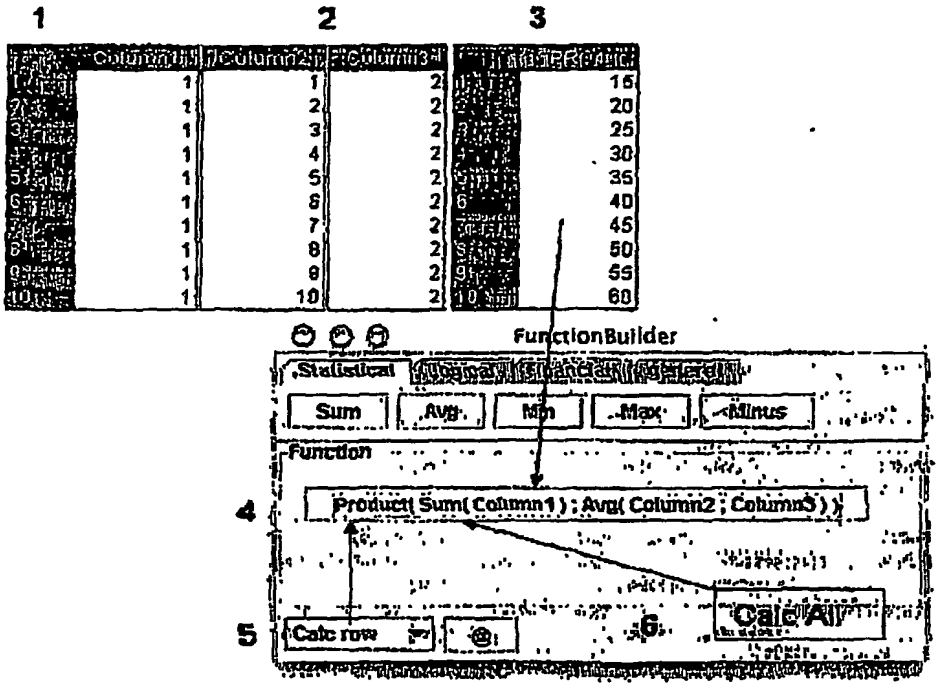

1
2
3
Column1
Column2
Column3
FunctionBuilder
Statistical
Sum
Avg
Min
Max
Minus
Function
Product(Sum(Column1) ; Avg(Column2 ; Column3))
4
5
Calc row
6
Calc All

Fig. 23

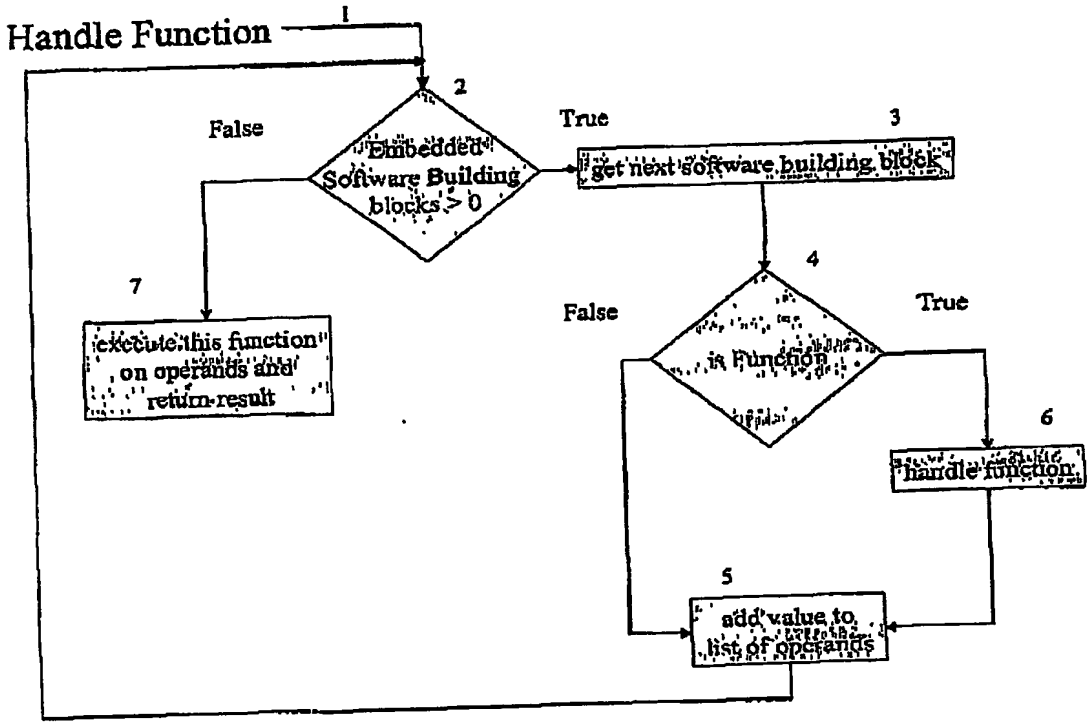

Handle Function
1
2
Embedded Software Building blocks > 0
False
True
3
get next software building block
4
is Function
False
True
5
add value to list of operands
6
handle function
7
execute this function on operands and return result

Fig. 24

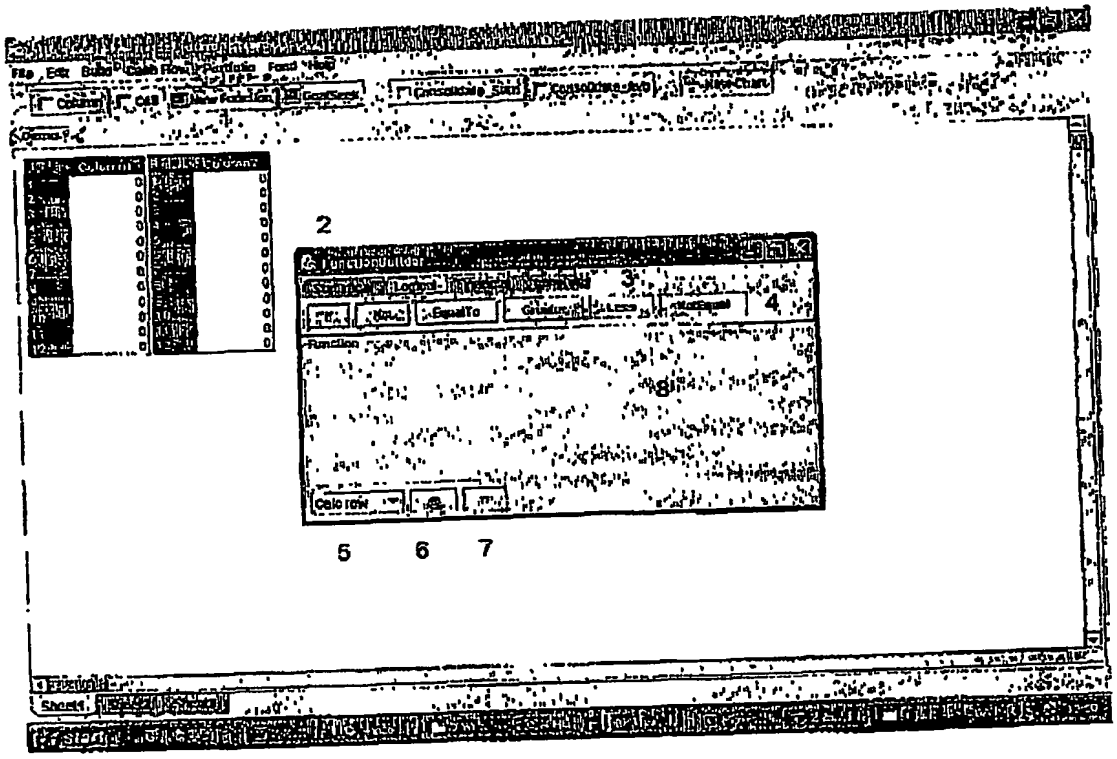


Fig. 26

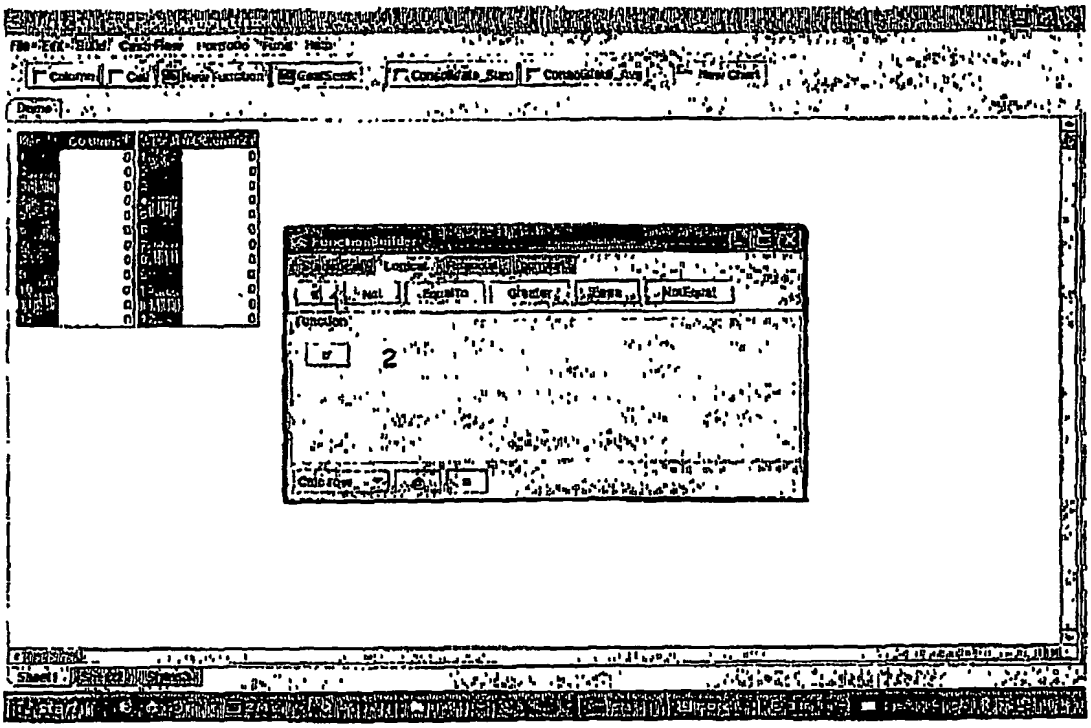


Fig. 27

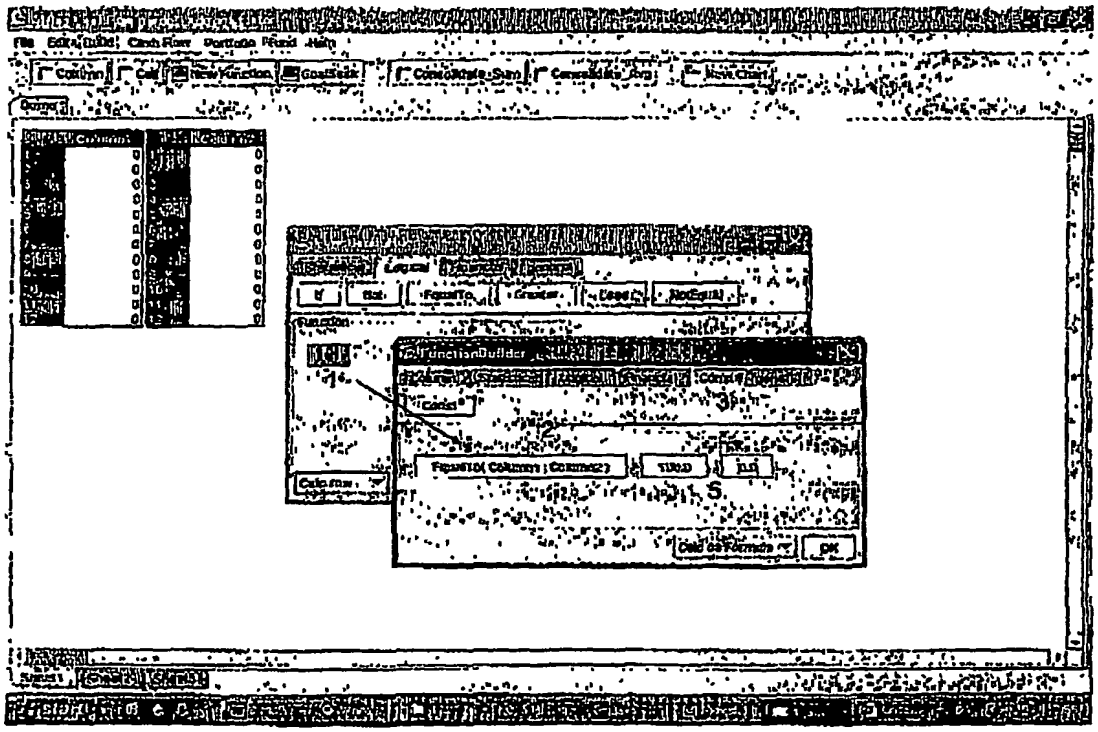


Fig. 28

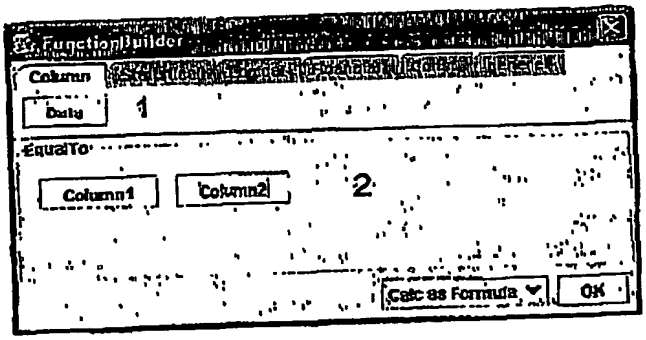

FunctionBuilder
Column
Data
1
EqualTo
Column1
Column2
2
Calc as Formula
OK

Fig. 29

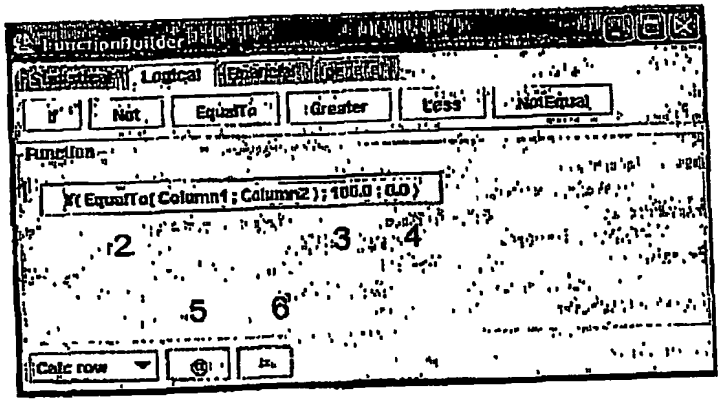

FunctionBuilder
Logical
If
Not
EqualTo
Greater
Less
NotEqual
Function
If(EqualTo(Column1 ; Column2) ; 100.0 ; 0.0)
2
3
4
5
6
Calc row

Fig. 30

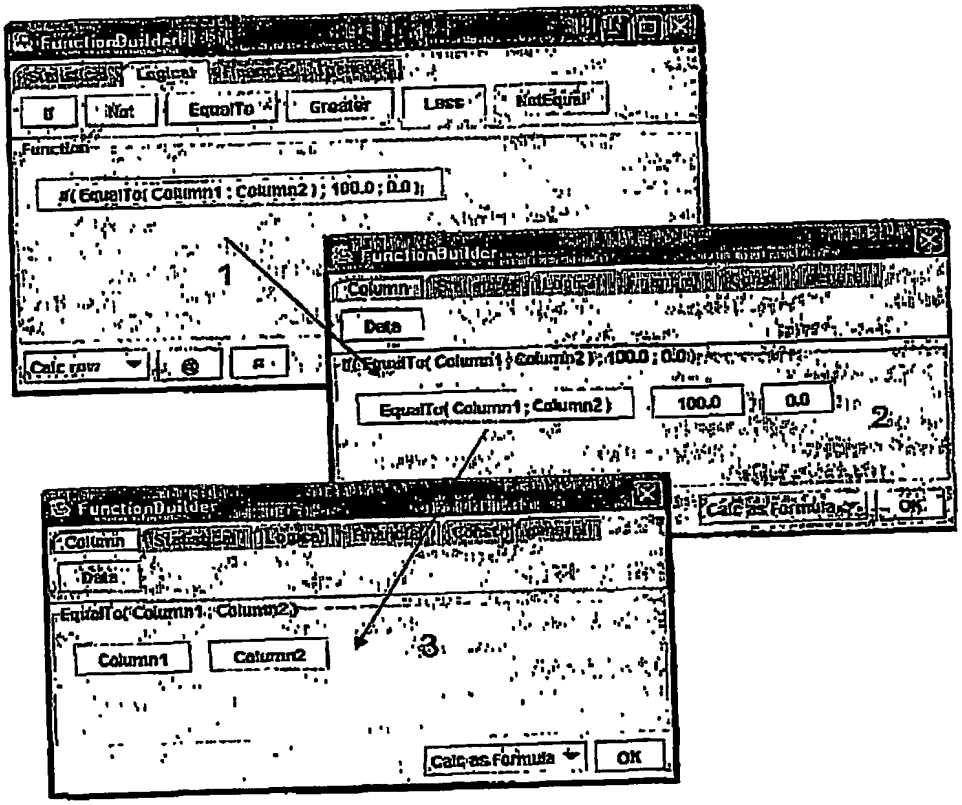

FunctionBuilder
Logical
If
Not
EqualTo
Greater
Less
NotEqual
Function
If(EqualTo(Column1 ; Column2) ; 100.0 ; 0.0);
1
Calc.you
FunctionBuilder
Column
Data
If(EqualTo(Column1 ; Column2) ; 100.0 ; 0.0);
EqualTo(Column1 ; Column2)
100.0
0.0
2
Calc as Formula
OK
FunctionBuilder
Column
Data
EqualTo(Column1 ; Column2)
Column1
Column2
3
Calc as Formula
OK

Fig. 31

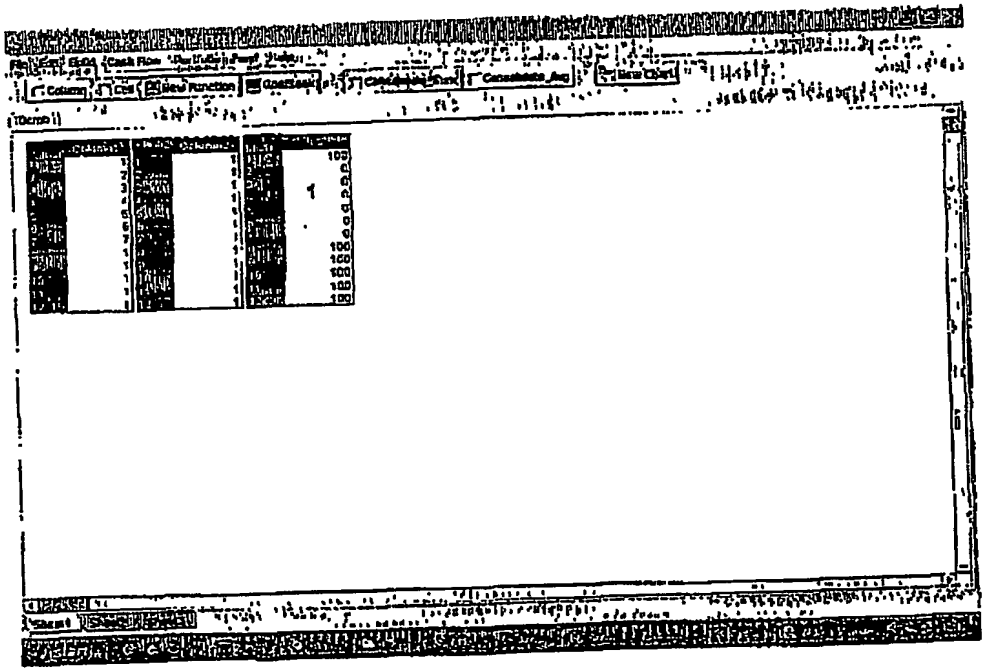


Fig. 32a

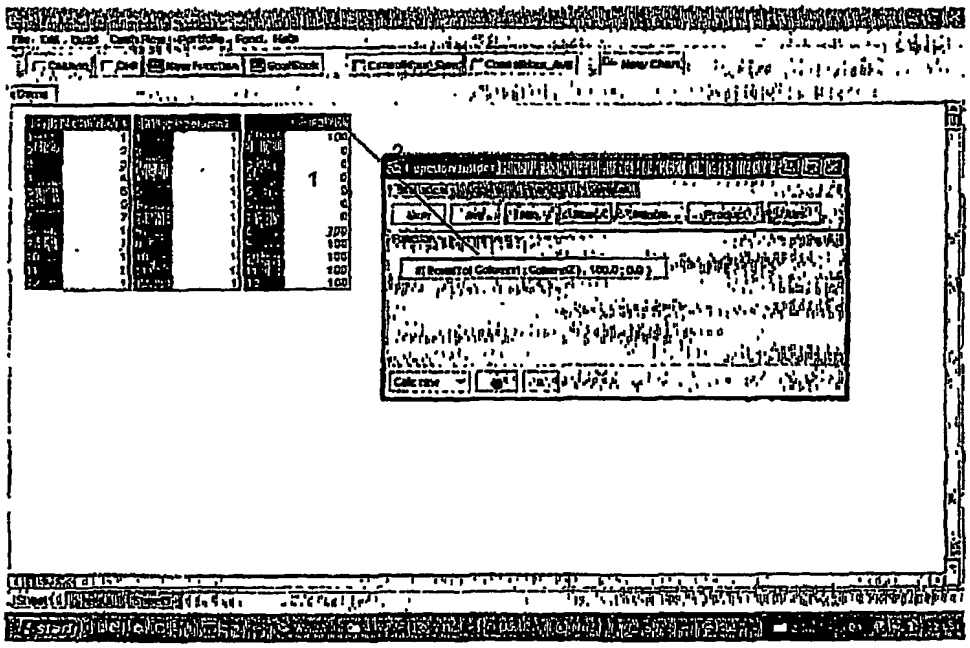


Fig. 32b

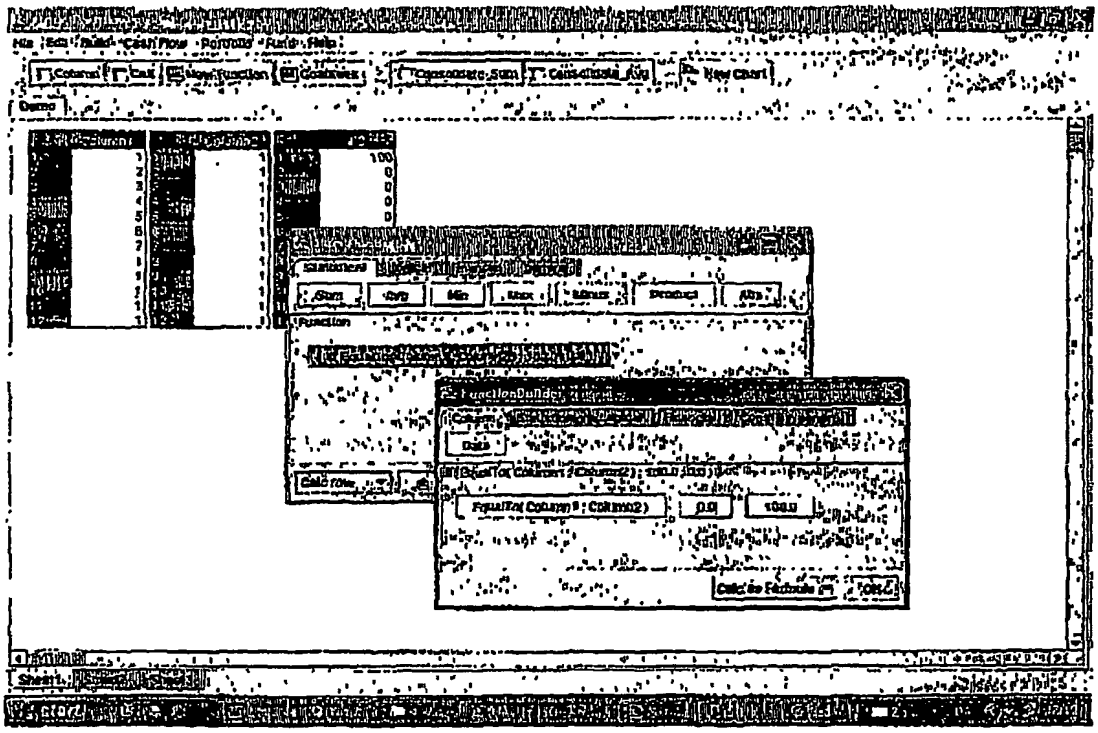


Fig. 33a

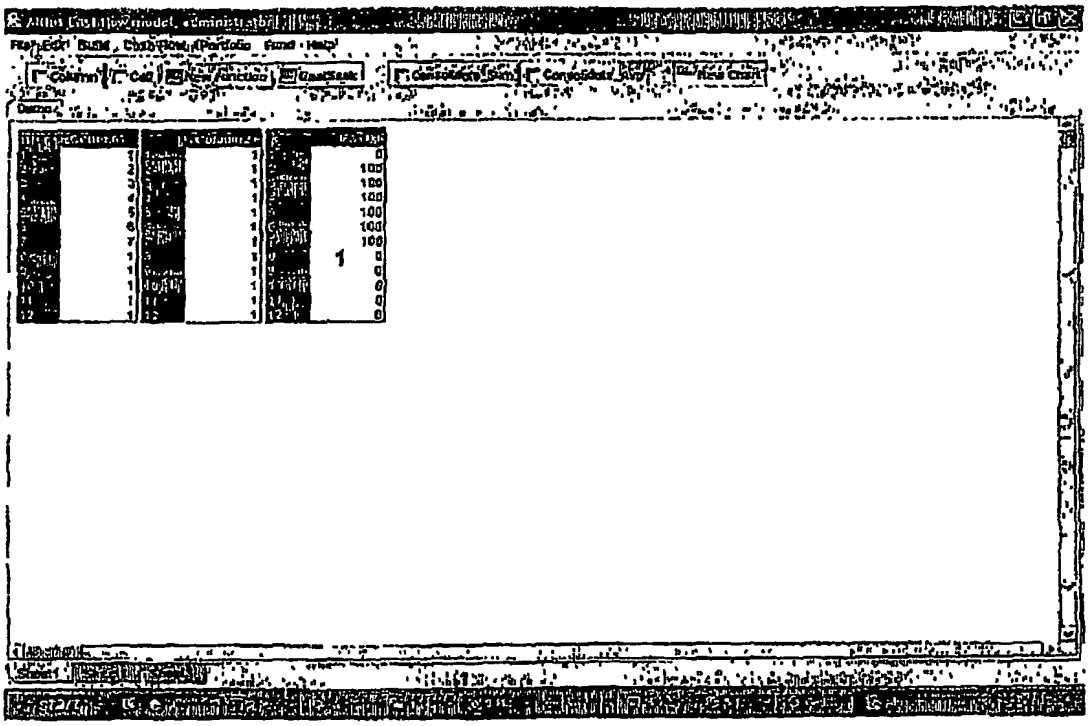


Fig. 33b

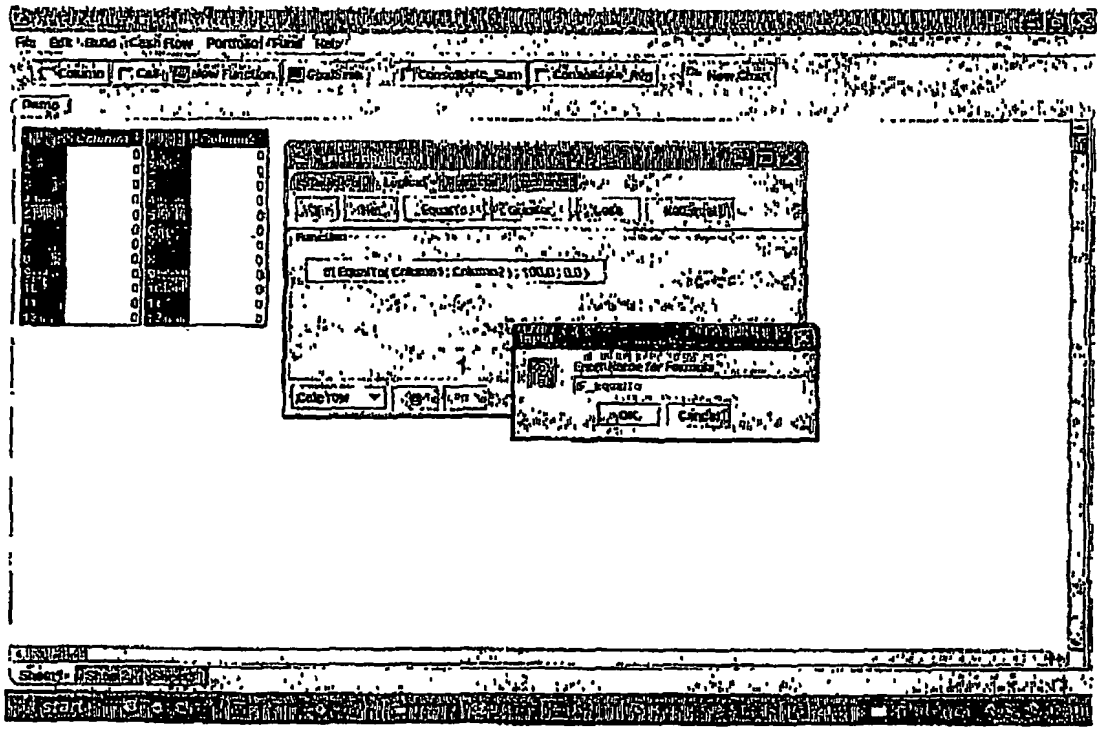


Fig. 34

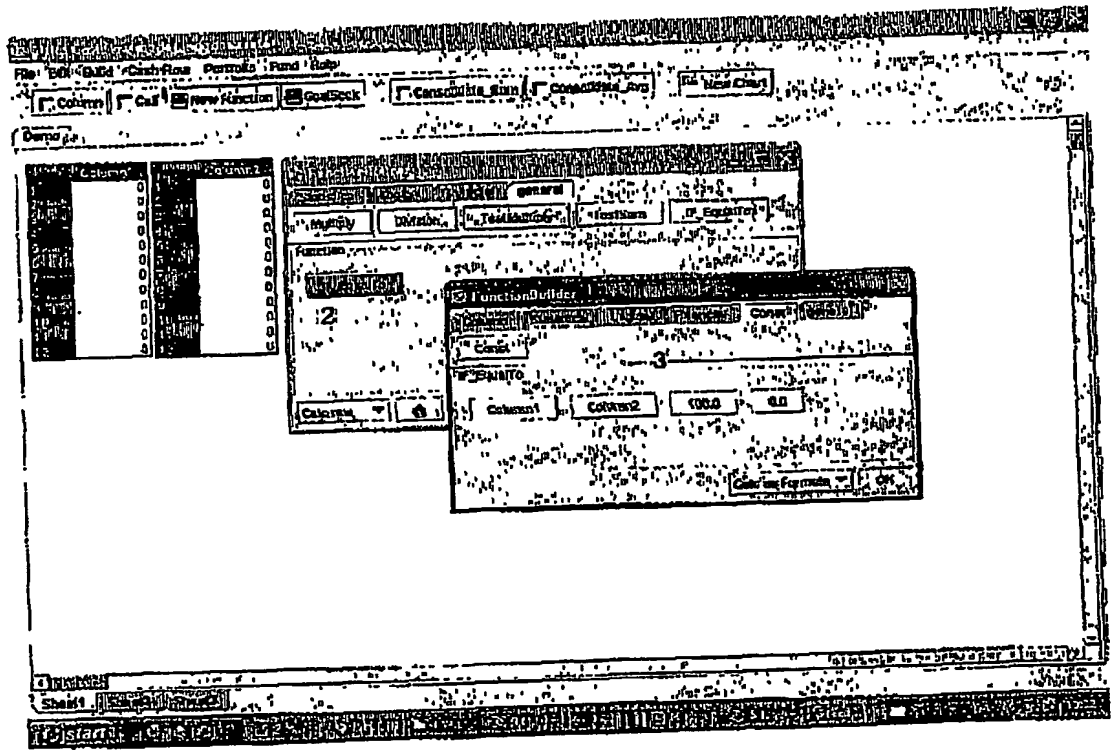

File Edit Build Cash Flow Periods Fund Help
Column
Call
new Function
GoalSeek
Consolidate Sum
Consolidate Avg
New Chart
Demo
Column1
Column2
general
Function
FunctionBuilder
Clear
Build To
Column1
Column2
100.0
0.0
OK
Sheet1
start

Fig. 35

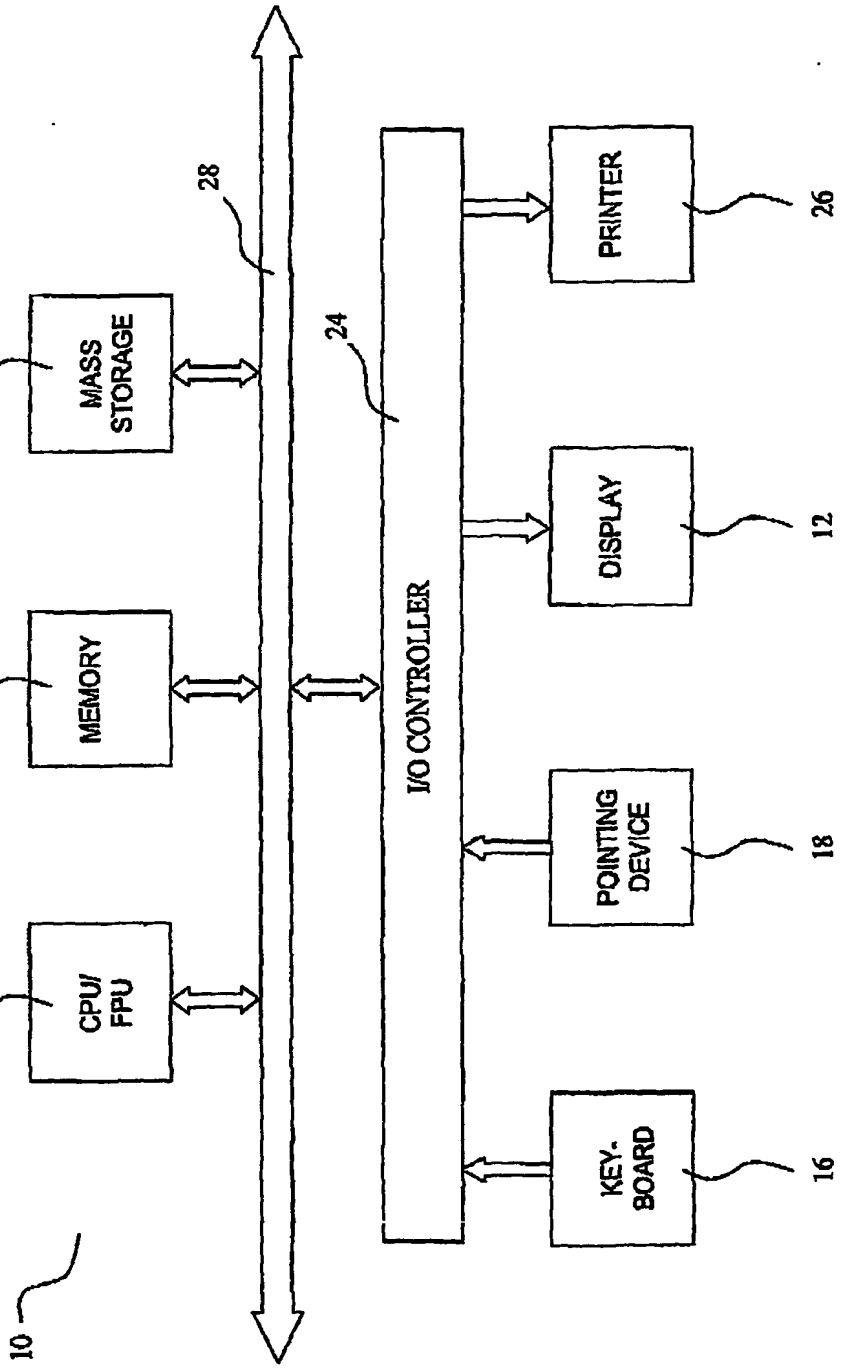

10
28
22
MASS STORAGE
20
MEMORY
14
CPU/ FPU
24
I/O CONTROLLER
PRINTER
26
DISPLAY
12
POINTING DEVICE
18
KEY-BOARD
16

Fig. 36

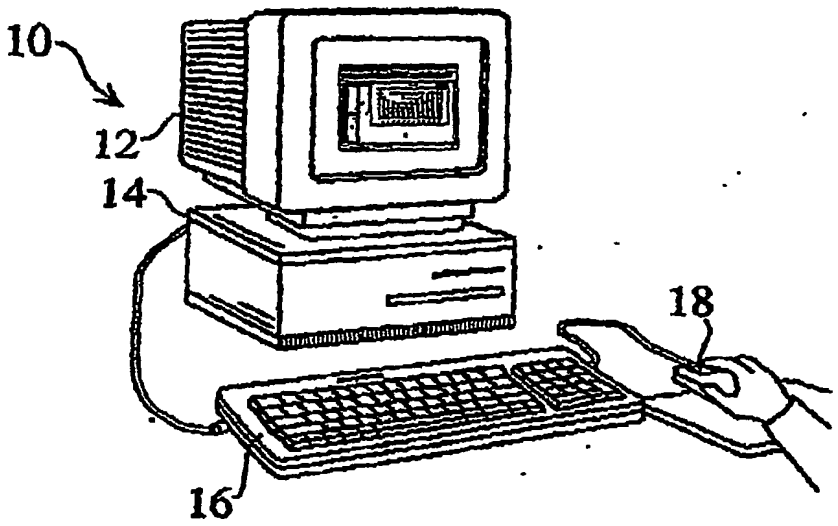


Fig. 37

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☑ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☑ FADED TEXT OR DRAWING
- ☑ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☑ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☑ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.